Higher-Order Linearisability

Andrzej S. Murawski[a], Nikos Tzevelekos[b,1]

[a] *University of Oxford*
[b] *Queen Mary University of London*

Abstract

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

Keywords: Linearisability, Concurrency, Higher-Order Computation

1. Introduction

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

In the concurrent setting, a notion that has been particularly influential is that of *linearisability* [1]. Linearisability requires that, for each history generated by a library, one should be able to find another history from the specification (a *linearisation*), which matches the former up to certain rearrangements of events. In the original formulation by Herlihy and Wing [1], these permutations were not allowed to disturb the order between library returns and client calls. Moreover, linearisations were required to be *sequential* traces, that is, sequences of method calls immediately followed by their returns.

In this paper we shall work with *open higher-order* libraries, which provide implementations of *public* methods and may themselves depend on *abstract* ones, to be supplied by parameter libraries. The classic notion of linearisability only applies to closed libraries (without abstract methods). Additionally, both method arguments and results had to be of *ground* type. The closedness limitation was recently lifted in [2, 3], which distinguished between public (or *implemented*) and abstract methods (*callable*). Although [2] did not in principle exclude higher-order functions, those works focussed on linearisability for the case where the allowable methods were restricted to first-order

[1] Research supported by EPSRC (EP/P004172/1).

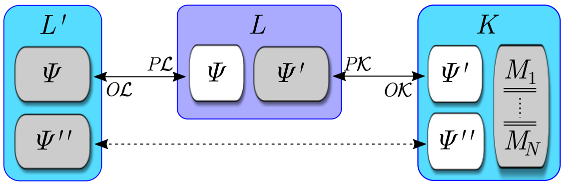

Figure 1: A library $L : \Psi \to \Psi'$ in environment comprising a parameter library $L' : \varnothing \to \Psi, \Psi''$ and a client K of the form $\Psi', \Psi'' \vdash M_1 \| \cdots \| M_N$.

functions ($\mathsf{int} \to \mathsf{int}$). Herein, we give a systematic exposition of linearisability for general higher-order concurrent libraries, where methods can be of arbitrary higher-order types. In doing so, we also propose a corresponding notion of sequential history for higher-order library interactions.

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

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

Our framework is operational in flavour and draws upon concurrent [4, 5] and operational game semantics [6, 7, 8]. We shall model library use as a game between two participants: *Player (P)*, corresponding to the library L, and *Opponent (O)*, representing the environment (L', K) in which the library was deployed. Each call will be of the form $\mathsf{call}\, m(v)$ with the corresponding return of the shape $\mathsf{ret}\, m(v)$, where v is a value. As we work in a higher-order framework, v may contain functions, which can participate in subsequent calls and returns. Histories will be sequences of *moves*, which are calls and returns paired with thread identifiers. A history is sequential just if every move produced by O is immediately followed by a move by P in the same thread. In other words, the library immediately responds to each call or return delivered by the environment. In contrast to classic linearisability, the move by O and its response by P need not be a call/return pair, as the higher-order setting provides more possibilities (in particular, the P response may well be a call). Accordingly, linearisable higher-order histories can be seen as sequences of atomic segments (linearisation points), starting at environment moves and ending with corresponding library moves.

In the spirit of [3], we are going to consider two scenarios: one in which K and L' share an explicit communication channel (the general case) as well as a situation in which they can only communicate through the library (the encapsulated case). Further, we also handle the case in which extra closure assumptions can be made about the parameter library (the relational case), which can be useful for dealing with a variety of assumptions on the use of parameter libraries that may arise in practice. In each case, we present a candidate definition of linearisability and illustrate it with tailored examples. The suitability of each kind of linearisability is demonstrated by showing that it implies the relevant form of contextual approximation (refinement). We also examine compositionality of the proposed concepts. One of our examples will discuss the implementation of the flat-combining approach [9, 3], adapted to higher-order types.

```
1   public count, update;
2   Lock lock;
3   F := λx.0;
4
5   count = λi. (!F)i
6   update = λ(i, g). aux(i,g,count i)
7
8   aux = λ(i, g, j).
9     let y = |g j| in
10      lock.acquire();
11      let f = !F in
12        if (j == (f i)) then (
13          F := λx. if (x == i) then y
14                       else (f x) ;
15          lock.release();
16          y )
17        else (
18          lock.release();
19          aux(i,g,f i) )
```

```
1   public count, update, reset;
2   abstract default;
3   Lock lock;
4   F := λx.0;
5   count = λi. (!F)i
6   update = λ(i, g). aux(i,g,count i)
7   ...
20  reset = λi.
21    lock.acquire();
22    let y = |default i| in
23      let f = !F in
24        F := λx. if (x == i) then y
25                     else (f x);
26        lock.release();
27        y
```

Figure 2: Left: Multiset library L_{mset} with public methods *count* : int → int and *update* : int × (int → int) → int. Right: Parameterised multiset library L_{mset2} (lines 8-19 as in LHS) with public methods *count*, *reset* : int → int, *update*: int × (int → int) → int; abstract method *default* : int → int.

The paper is an extended version of [10] and contains complete proofs, fully elaborated examples and appendices with further technical material, e.g. on compositionality.

Example: a higher-order multiset library

Higher-order libraries are common in languages like ML, Java, Python, etc. As an illustrative example, we consider a library written in ML-like syntax which implements a multiset data structure with integer elements. For simplicity, we assume that its signature contains just two methods:

$$count : \mathsf{int} \to \mathsf{int}, \quad update : (\mathsf{int} \times (\mathsf{int} \to \mathsf{int})) \to \mathsf{int}\,.$$

The former method returns for each integer its multiplicity in the multiset – this is 0 if the integer is not a member of the multiset. On the other hand, *update* takes as an argument an integer i and a function g, and updates the multiplicity j of i in the multiset to $|g(j)|$ (we use the absolute value of $g(j)$ in order to meet the multiset requirement that element multiplicities not be negative; alternatively, we could have used exceptions to quarantine such client method behaviour). Methods with the same functionalities can be found in the multiset module of the ocaml-containers library [11]. While our example is simple, the same kind of analysis as below can be applied to more intricate examples such as *map* methods for integer-valued arrays, maps or multisets.

Example 1 (Multiset). Consider the concurrent multiset library L_{mset} in Figure 2 on the LHS (the RHS will be discussed only later). It uses a private reference for storing the multiset's characteristic function and reads *optimistically*, without locking (cf. [12, 13]). The *update* method in particular reads the current multiplicity of the given element i (via *count*) and computes its new multiplicity without acquiring a lock on the characteristic function. It only acquires a lock when it is ready to write the new value (line 10) in the

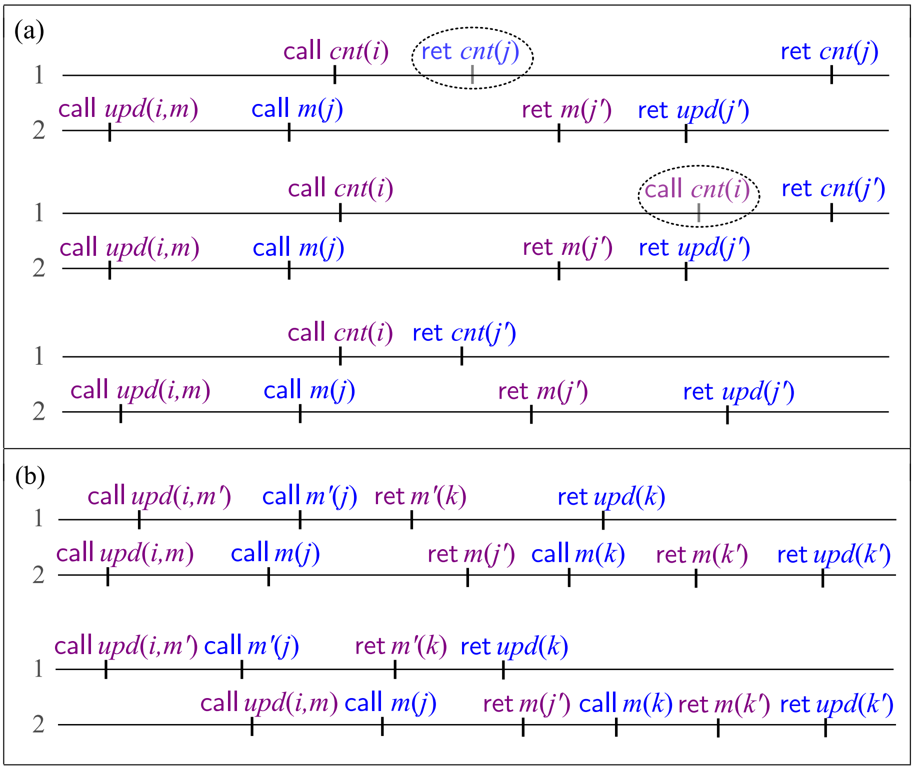

Figure 3: Example histories of L_{mset}.

hope that the value at i will still be the same and the update can proceed; if not, another attempt to update the value is made.

Let us look at some example executions of the library via their resulting histories, i.e. sequences of method calls and returns between the library and a client. In the topmost block (a) of history diagrams of Figure 3, we see three such executions. Note that we do not record internal calls to *count* or *aux*, and use m and variants for method identifiers (names). We use the abbreviation *cnt* for *count*, and *upd* for *update*, and initially ignore the circled events for *cnt*. Each execution involves 2 threads.

In the first execution, the client calls *update*(i, m) in the second thread, and subsequently calls *count*(i) in the first thread. The code for *update* stipulates that first *count*(i) be called internally, returning some multiplicity j for i, and then $m(j)$ should be called. As soon m returns a value j', *update* sets the multiplicity of i to j' and itself returns j'. The last event in this history is a return of *count* in the first thread with the old value j. According to our proposed definition, this history will be linearisable to another, intuitively correct one: the last return can be moved to the circled position. At this point the notion of linearisability is used informally, but it will be made precise in the following sections. In the second execution, the last return of *count* in the first thread returns the updated value. In this case, we will be able to move call *cnt*(i) to the circled position to obtain a linearisation, which is obviously correct. Finally, in the third execution we have a history that will turn out non-linearisable to an intuitively correct history. Indeed, we should not be able to return the updated value in the first thread before m has returned it in the second one.

The two histories in block (b) in the same figure demonstrate the mechanism for updates. The first history will be linearisable to the second one. In the second history we see that both threads try to update the same element i, but the first one succeeds in it first and returns k on *update*. Then, the second thread realises that the value of i has been updated to k and calls m again, this time with argument k. An important feature of the second history is that it is *sequential*: each client event (call or return) is immediately followed by a library event.

Observe that the rearrangements discussed above involve either advancing a library action or postponing an environment action and that each action could be a call or a return. Definition 9 will capture this formally. For now, we note that this generalises the classic setting [1], where library method returns could be advanced and environment method calls deferred.

The next section will introduce histories along with the proposed notion of linearisability. In Section 3 we present the syntax for libraries and clients, and in Section 3.1 we define their semantics in terms of histories and co-histories respectively.

2. Higher-order linearisability

We examine higher-order libraries interacting with their context by means of abstract and public methods. In particular, we shall rely on types given by the grammar below. We let Meths stand for the set of *method names* and assume $\mathsf{Meths} = \biguplus_{\theta,\theta'} \mathsf{Meths}_{\theta,\theta'}$, where each set $\mathsf{Meths}_{\theta,\theta'}$ contains names for methods of type $\theta \to \theta'$. Methods are ranged over by m (and variants). We let v range over computational *values*, which include a unit value, integers, methods and pairs of values.

$$\theta ::= \mathsf{unit} \mid \mathsf{int} \mid \theta \times \theta \mid \theta \to \theta \qquad\qquad v ::= () \mid i \mid m \mid (v, v)$$

The framework of a higher-order library and its environment is depicted in Figure 1. Given $\Psi, \Psi' \subseteq \mathsf{Meths}$, a library L is said to have type $\Psi \to \Psi'$ if it defines public methods with names (and types) as in Ψ', using abstract methods Ψ. The environment of L consists of a *client* K (which invokes public methods of Ψ'), and a *parameter library* L' (which provides code for the abstract methods Ψ). In general, K and L' may interact via a disjoint set of methods $\Psi'' \subseteq \mathsf{Meths}$, to which L has no access.

In the rest of this paper, we shall implicitly assume that we work with a library L operating in an environment presented in Figure 1. The client K will consist of a fixed number N of concurrent threads. Next we introduce a notion of history tailored to the setting and define how histories can be linearised.

2.1. Higher-order histories

The operational semantics of libraries will be given in terms of *histories*, which are sequences of method calls and returns, each decorated with a thread identifier t and a *polarity index* X, where $X \in \{O, P\}$, as shown below.

$$(t, \mathsf{call}\, m(v))_X \qquad (t, \mathsf{ret}\, m(v))_X$$

We shall refer such decorated calls and returns as ***moves***. Here, m is a method name and v is a value of a matching type. The index X specifies who produces the move: the library L (polarity P), or its environment (L', K) (polarity O). Using notation e.g. from [3], P corresponds to !, and O to ?. We may be dropping the polarity of a move when it is not important or no confusion arises by doing so.

The choice of indices is motivated by the fact that the moves can be seen as defining a 2-player game between the library (L), which represents the *Proponent* player in the game (P), and its environment (L', K) that represents the *Opponent* (O). Finally, we let the ***dual*** polarity of X be X', where $X \neq X'$.

Next we proceed to define histories. Their definition will rely on a more primitive concept of *prehistories*, which are sequences of O/P-indexed method calls and returns that respect a stack discipline.

Definition 2. *Prehistories* are sequences generated by one of the grammars:

$$\begin{array}{lcl} \mathsf{PreH}_O & ::= & \epsilon \mid \mathsf{call}\, m(v)_O\ \mathsf{PreH}_P\ \mathsf{ret}\, m(v')_P\ \mathsf{PreH}_O \\ \mathsf{PreH}_P & ::= & \epsilon \mid \mathsf{call}\, m(v)_P\ \mathsf{PreH}_O\ \mathsf{ret}\, m(v')_O\ \mathsf{PreH}_P \end{array}$$

where, if $m \in \mathsf{Meths}_{\theta,\theta'}$, the types of v, v' must match θ, θ' respectively. We let $\mathsf{PreH} = \mathsf{PreH}_O \cup \mathsf{PreH}_P$.

Thus, prehistories from PreH_O start with an O-call, while those in PreH_P start with a P-call. In each case, the polarities inside a prehistory alternate between O and P, and the polarities of calls and matching returns are always dual (*returns dual to calls*).

Histories will be interleavings of prehistories tagged with thread identifiers (natural numbers), subject to a set of well-formedness constrains. In particular, a history h for library $L : \Psi \to \Psi'$ will have to begin with an O-move and satisfy the following conditions, to be formalised in Definition 3.

1. The name of any method called in h must come from Ψ or Ψ', or be introduced earlier in h as a higher-order argument or result (*no methods out of thin air*). In addition:
   - if the method is from Ψ', the call must be tagged with O (i.e. issued by K);
   - if the method is from Ψ, the call must be tagged with P (i.e. issued by L towards L');
   - for a call of method $m \notin \Psi \cup \Psi'$ to be valid, m must be introduced in an earlier move of dual polarity (*calls dual to introductions*).
2. Any method name appearing inside a call or return argument in h must be *fresh*, i.e. not used earlier (*introductions always fresh*).
   - This reflects the assumption that methods can be called and returned from, but not compared for identity equality. It is therefore a requirement towards the *completeness* of histories as a semantics for concurrent libraries. For example, this ensures that rules like η-equality are preserved in the semantics.
   - The condition serves the additional purpose of making the setting described in Figure 1 robust, as it prevents method names in Ψ from being leaked to the client K. This ensures that encapsulation cannot be broken.

Given $h \in \mathsf{PreH}$ and $t \in \mathbb{N}$, we write $t \times h$ for h in which each element is decorated with t:

$$t \times ((x_1)_{X_1}(x_2)_{X_2}\cdots(x_k)_{X_k}) = (t,x_1)_{X_1}(t,x_2)_{X_2}\cdots(t,x_k)_{X_k}.$$

We say that a move $(t,x)_X$ ***introduces*** a name $m \in \mathsf{Meths}$ when $x \in \{\mathsf{call}\, m'(v), \mathsf{ret}\, m'(v)\}$ for some m', v such that v contains m.

Definition 3. Given Ψ, Ψ', the set of ***histories*** over $\Psi \to \Psi'$, written $\mathcal{H}_{\Psi,\Psi'}$, is defined by

$$\mathcal{H}_{\Psi,\Psi'} = \bigcup_{N>0} \bigcup_{h_1,\cdots,h_N \in \mathsf{PreH}_O} (1 \times h_1) \mid \cdots \mid (N \times h_N)$$

where $(1 \times h_1) \mid \cdots \mid (N \times h_N)$ is the set of all interleavings of $(1 \times h_1), \cdots, (N \times h_N)$ satisfying:

1. For any $s_1\,(t, \mathsf{call}\, m(v))_X\, s_2 \in \mathcal{H}_{\Psi,\Psi'}$, either $m \in \Psi'$ and $X = O$, or $m \in \Psi$ and $X = P$, or there is a move $(t', x)_{X'}$ in s_1 that introduces m and $X \neq X'$.
2. For any $s_1 (t, x)_X s_2 \in \mathcal{H}_{\Psi,\Psi'}$ and any m, if m is introduced by x then m must not occur in s_1.

Note that the definition supports scenarios in which a method sent as a parameter by one thread can be called by a different thread. This feature will be explored in Example 18.

A history $h \in \mathcal{H}_{\Psi,\Psi'}$ is called ***sequential*** if it is of the form

$$h = (t_1, x_1)_O (t_1, x_1')_P \cdots (t_k, x_k)_O (t_k, x_k')_P$$

for some t_i, x_i, x_i'. We write $\mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$ for the set of all sequential histories from $\mathcal{H}_{\Psi,\Psi'}$.

We shall range over $\mathcal{H}_{\Psi,\Psi'}$ using h, s (and variants). The subscripts Ψ, Ψ' will often be omitted. Given a history h, we shall write $\overline{h}$ for the sequence of moves obtained from h by dualising all move polarities inside it. The set of ***co-histories*** over $\Psi \to \Psi'$ will be $\mathcal{H}^{co}_{\Psi,\Psi'} = \{\overline{h} \mid h \in \mathcal{H}_{\Psi,\Psi'}\}$.

While in this section histories will be extracted from example libraries informally, in Section 3.1 we give the formal semantics $[\![L]\!]$ of libraries. For each $L : \Psi \to \Psi'$, we shall have $[\![L]\!] \subseteq \mathcal{H}_{\Psi,\Psi'}$.

Remark 4. The notion of history introduced above extends the classic notion from [1] to higher-order types. It also extends the notion presented in [3]. The intuition behind the definition is that a history is a sequence of (well-bracketed) method calls and returns, called *moves*, each tagged with a thread identifier and a polarity, where polarities track the originators and recipients of moves. Moves may be calls or returns related to methods given in the library interface ($\Psi \to \Psi'$), or dynamically created methods that appear earlier inside the histories – recall that, in a higher-order setting, methods can be passed around as arguments to calls or be returned as results by other methods. On the other hand, a sequential history is one in which the operations performed by the library can be perceived as **atomic**, that is, each move produced by O is to be immediately followed by the library's response, which is a P move in the same thread.

Example 5 (Multiset spec). We now revisit our first example and provide a specification for it. Recall the multiset library L_{mset} from Figure 2. Our verification goal will be to prove linearisability of L_{mset} to a specification $A_{\mathsf{mset}} \subseteq \mathcal{H}^{\mathsf{seq}}_{\varnothing,\Psi}$, where $\Psi = \{\mathit{count}, \mathit{update}\}$, which we define below. Intuitively, the specification stipulates that the multiset operations are functionally correct and only includes sequential histories. For example, the following histories are in the specification:

$$(1, \mathsf{call}\, \mathit{upd}(5, m))_O\, (1, \mathsf{call}\, m(5))_P\, (1, \mathsf{call}\, \mathit{cnt}(5))_O\, (1, \mathsf{ret}\, \mathit{cnt}(0))_P$$
$$(1, \mathsf{ret}\, m(42))_O\, (1, \mathsf{ret}\, \mathit{upd}(42))_P$$
$$(1, \mathsf{call}\, \mathit{upd}(5, m))_O\, (1, \mathsf{call}\, m(5))_P\, (2, \mathsf{call}\, \mathit{upd}(5, m'))_O\, (2, \mathsf{call}\, m'(5))_P$$
$$(1, \mathsf{ret}\, m(42))_O\, (1, \mathsf{ret}\, \mathit{upd}(42))_P\, (3, \mathsf{call}\, \mathit{cnt}(5))_O\, (3, \mathsf{ret}\, \mathit{cnt}(42))_P$$
$$(2, \mathsf{ret}\, m'(24))_O\, (2, \mathsf{ret}\, \mathit{upd}(24))_P\, (1, \mathsf{call}\, \mathit{cnt}(5))_O\, (1, \mathsf{ret}\, \mathit{cnt}(24))_P$$

while the next ones are not:

$$(1, \mathsf{call}\; upd(5,m))_O\,(1, \mathsf{call}\; m(5))_P\,(1, \mathsf{call}\; cnt(5))_O\,(1, \mathsf{ret}\; cnt(42))_P \cdots$$
$$(1, \mathsf{call}\; upd(5,m))_O\,(2, \mathsf{call}\; upd(6,m'))_O \cdots$$
$$(1, \mathsf{call}\; upd(5,m))_O(1, \mathsf{call}\; m(5))_P(2, \mathsf{call}\; upd(5,m'))_O(2, \mathsf{call}\; m'(5))_P$$
$$(1, \mathsf{ret}\; m(42))_O(1, \mathsf{ret}\; upd(42))_P(3, \mathsf{call}\; cnt(5))_O(3, \mathsf{ret}\; cnt(42))_P$$
$$(2, \mathsf{ret}\; m'(24))_O(2, \mathsf{ret}\; upd(24))_P(1, \mathsf{call}\; cnt(5))_O(1, \mathsf{ret}\; cnt(42))_P$$

A_{mset} will certify that L_{mset} correctly implements some integer multiset I whose elements change over time according to the moves in h. For a multiset I and natural numbers i, j, we write $I(i)$ for the multiplicity of i in I, and $I[i \mapsto j]$ for I with its multiplicity of i set to j. We shall stipulate that moves inside histories $h \in A_{\mathsf{mset}}$ be annotatable with multisets I in such a way that the multiset is empty at the start of h (i.e. $I(i) = 0$ for all i) and:

- If I is changed between two consecutive moves in h then the second move is a P-move. In other words, the client cannot directly update the elements of I.
- Each call to *count* on argument i must be immediately followed by a return with value $I(i)$, and with I remaining unchanged.
- Each call to *update* on (i, m) must be followed by a call to m on $I(i)$, with I unchanged. Moreover, m must later return with some value j. Assuming at that point the multiset will have value J, if $I(i) = J(i)$ then the next move is a return of the original *update* call, with value j; otherwise, a new call to m on $J(i)$ is produced, and so on.

We formally define the specification next.

Let $\mathcal{H}^{\circ}_{\varnothing,\Psi}$ contain sequences of moves from $\varnothing \to \Psi$ accompanied by a multiset (i.e. the sequences consist of elements of the form $(t, x, I)_X$). For each $s \in \mathcal{H}^{\circ}_{\varnothing,\Psi}$, we let $\pi_1(s)$ be the history extracted by projection, i.e. $\pi_1(s) \in \mathcal{H}_{\varnothing,\Psi}$. For each t, we let $s \upharpoonright t$ be the subsequence of s of elements with first component t. Writing $\sqsubseteq_{\mathrm{pre}}$ for the prefix relation, we define $A_{\mathsf{mset}} = \{\, \pi_1(s) \mid s \in A^{\circ}_{\mathsf{mset}} \,\}$ where:

$$A^{\circ}_{\mathsf{mset}} = \{\, s \in \mathcal{H}^{\circ}_{\varnothing,\Psi} \mid \pi_1(s) \in \mathcal{H}^{\mathsf{seq}}_{\varnothing,\Psi} \wedge \forall t.\, s \upharpoonright t \in \mathcal{S} \wedge s = (\_, I)_O\, s' \implies \forall i. I(i) = 0 \\ \wedge\ \forall s'(\_, I)_P(\_, J)_O \sqsubseteq_{\mathrm{pre}} s.\ I = J \,\}$$

and, for each t, the set of t-indexed annotated histories $\mathcal{S}$ is given by the following grammar:

$$\begin{aligned}
\mathcal{S} &\to \epsilon \mid (t, \mathsf{call}\; cnt(i), I)_O\,(t, \mathsf{ret}\; cnt(I(i)), I)_P\,\mathcal{S} \\
&\quad\ \mid (t, \mathsf{call}\; upd(i,m), I)_O\,\mathcal{M}^{i,j}_{I,J}\,(t, \mathsf{ret}\; upd(|j|), J[i \mapsto |j|])_P\,\mathcal{S} \\
\mathcal{M}^{i,j}_{I,J} &\to (t, \mathsf{call}\; m(I(i)), I)_P\,\mathcal{S}\,(t, \mathsf{ret}\; m(j), J)_O \qquad \text{provided } J(i) = I(i) \\
\mathcal{M}^{i,j}_{I,J} &\to (t, \mathsf{call}\; m(I(i)), I)_P\,\mathcal{S}\,(t, \mathsf{ret}\; m(j'), J')_O\,\mathcal{M}^{i,j}_{J',J} \qquad \text{provided } J'(i) \neq I(i)
\end{aligned}$$

By definition, all histories in A_{mset} are sequential. The elements of $A^{\circ}_{\mathsf{mset}}$ carry along the multiset I that is being represented. The conditions on $A^{\circ}_{\mathsf{mset}}$ stipulate that I is initially empty and that O cannot change the value of I, while the rest of the conditions above are imposed by the grammar for $\mathcal{S}$. With the notion of linearisability to be introduced next, we will be able to show that $[\![L_{\mathsf{mset}}]\!]$ is indeed linearisable to A_{mset}.

Remark 6. In our framework (higher-order computation with state) specifications are necessarily close to implementations. For example, they need to preserve the exact number of calls/returns, because each of them could trigger a potential side effect. As in [1], specifications contain sequential histories.

2.2. *Three notions of linearisability*

We present three notions of linearisability. First introduce a general notion that generalises classic linearisability [1] and parameterised linearisability [3]. We then develop two more specialised variants: a notion of encapsulated linearisability, following [3], that captures scenarios where the parameter library and the client cannot directly interact; and a relational notion whereby context behaviour (client and parameter library) is known to be relationally invariant.

2.2.1. *General linearisability*

We begin by introducing a class of reorderings on histories.

Definition 7. Let $\triangleleft_{PO} \subseteq \mathcal{H}_{\Psi,\Psi'} \times \mathcal{H}_{\Psi,\Psi'}$ be the smallest binary relation over $\mathcal{H}_{\Psi,\Psi'}$ satisfying, for any $t \neq t'$:

$$s_1\,(t',x')_{Z'}\,(t,x)_Z\,s_2 \;\triangleleft_{PO}\; s_1\,(t,x)_Z\,(t',x')_{Z'}\,s_2$$

whenever $Z = P$ or $Z' = O$.

Intuitively, two histories h_1, h_2 are related by $\triangleleft_{PO}$ if the latter can be obtained from the former by swapping two adjacent moves from different threads in such a way that, after the swap, a P-move will occur earlier or an O-move will occur later. Note that the relation always applies to adjacent moves of the same polarity. On the other hand, we do *not* have $s_1(t,x)_P(t',x')_O s_2 \triangleleft_{PO} s_1(t',x')_O(t,x)_P s_2$.

Example 8. Let $\Psi = \{m : \mathsf{int} \to \mathsf{int}\}$ and $\Psi' = \{m' : \mathsf{int} \to \mathsf{int}\}$. Consider $h, h' \in \mathcal{H}_{\Psi,\Psi'}$ given below.

$$\begin{aligned}
h = {} & (1, \mathsf{call}\, m(1))_O\,(2, \mathsf{call}\, m(5))_O\,(1, \mathsf{call}\, m'(2))_P\,(1, \mathsf{ret}\, m'(3))_O \\
& (2, \mathsf{call}\, m'(6))_P\,(2, \mathsf{ret}\, m'(7))_O\,(2, \mathsf{ret}\, m(8))_P\,(1, \mathsf{ret}\, m(4))_P \\
h' = {} & (1, \mathsf{call}\, m(1))_O\,(1, \mathsf{call}\, m'(2))_P\,(1, \mathsf{ret}\, m'(3))_O\,(1, \mathsf{ret}\, m(4))_P \\
& (2, \mathsf{call}\, m(5))_O\,(2, \mathsf{call}\, m'(6))_P\,(2, \mathsf{ret}\, m'(7))_O\,(2, \mathsf{ret}\, m(8))_P
\end{aligned}$$

Note that $h \triangleleft_{PO}^{*} h'$ by permuting $(2, \mathsf{call}\, m(5))_O$ rightwards and $(1, \mathsf{ret}\, m(4))_P$ leftwards.

As another example, we can revisit the histories in Figure 2. There, O-moves are coloured purple and P-moves are blue. In part (a) we can see that:

- the first history linearises to a sequential one by swapping a P-move of thread 1 to the left of two moves of thread 2,
- the second history linearises to a sequential one by swapping an O-move of thread 1 to the right of two moves of thread 2,
- the third history is already sequential and it cannot be linearised to a different one.

In part (b), on the other hand, the first history linearises to the second one by a series of swaps (left as exercise).

Analogously, one can consider the symmetric variant $\triangleleft_{OP}$ of $\triangleleft_{PO}$, which will turn out useful in our soundness argument.

Definition 9 (General Linearisability). Given $h_1, h_2 \in \mathcal{H}_{\Psi,\Psi'}$, we say that h_1 ***is linearised by*** h_2, written $h_1 \lhd h_2$, if $h_1 \lhd^*_{PO} h_2$.

Given libraries $L, L' : \Psi \to \Psi'$ and a set of sequential histories $A \subseteq \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$ we write $L \lhd A$, and say that L *can be linearised to* A, if for any $h \in [\![L]\!]$ there exists $h' \in A$ such that $h \lhd h'$. Moreover, we write $L \lhd L'$ if $L \lhd [\![L']\!] \cap \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$ (i.e. for all $h \in [\![L]\!]$ there is sequential $h' \in [\![L']\!]$ such that $h \lhd h'$).

Remark 10. The classic notion of linearisability from [1] states that h linearises to h' just if the return/call order of h is preserved in h' (and h' is sequential), i.e. if a return move precedes a call move in h then so is the case in h'. Observing that, in [1], return and call moves coincide with P- and O-moves respectively, we can see that our higher-order notion of linearisability is a generalisation of the classic notion.

Our definition shows that the ownership of actions is the key determinant of what moves can be swapped rather than the call/return distinction, which was prominent in the classic case. It just so happens that, for $\Psi = \varnothing$ and $\Psi' = \{m' : \mathsf{int} \to \mathsf{int}\}$, the two coincide.

For further comparison, recall that the classic definition allowed for $\mathsf{call}/\mathsf{call}$, $\mathsf{ret}/\mathsf{ret}$ and $\mathsf{call}/\mathsf{ret}$ swaps, but $\mathsf{ret}/\mathsf{call}$ was forbidden. According to our definition, what is allowed depends on polarity, so a $\mathsf{call}/\mathsf{call}$ swap may well be illegal if the first call is a P-move and the second call is an O-move. Similarly, a $\mathsf{ret}/\mathsf{call}$ swap is allowed as long as both actions belong to the same player or the return is an O-action and the call is a P-action. For instance, Example 8 involves the following kinds of swaps: $\mathsf{call}_O/\mathsf{call}_P$, $\mathsf{call}_O/\mathsf{ret}_O$, $\mathsf{ret}_P/\mathsf{ret}_P$, $\mathsf{ret}_O/\mathsf{ret}_P$, $\mathsf{call}_P/\mathsf{ret}_P$, $\mathsf{call}_O/\mathsf{ret}_P$.

Our emphasis on move ownership is motivated by Lemma 34, which will ultimately enable us to prove that, if $h \lhd h'$, then h' suffices to demonstrate the interactive potential of h. This intuition is formally captured in Theorem 35.

Remark 11. [3] defines linearisation using a "big-step" relation that applies a single permutation to the whole sequence. This contrasts with our definition as $\lhd^*_{PO}$, in which we combine multiple adjacent swaps. In Appendix A we show that the two definitions are equivalent.

2.2.2. *Encapsulated linearisability*

We next show that a more permissive notion of linearisability applies if the parameter library L' of Figure 1 is encapsulated, that is, the client K can have no direct access to it (i.e. $\Psi'' = \varnothing$). To capture this scenario, we define a second polarity function on moves, which determines the ***side*** of the move:

- a move with side $\mathcal{K}$ is played between the library L and the client K, while
- a move with side $\mathcal{L}$ is played between the library L and the parameter library L'.

Formally, given a history $h \in \mathcal{H}_{\Psi,\Psi'}$, we define a ***side*** function on its moves by:

$$\mathsf{side}((t, \mathsf{call}\, m(v))) = \begin{cases} \mathcal{K} & \text{if } m \in \Psi' \\ \mathcal{L} & \text{if } m \in \Psi \\ \mathsf{side}((t', x)) & \text{if } (t', x) \text{ introduces } m \end{cases}$$
$$\mathsf{side}((t, \mathsf{ret}\, m(v))) = \mathsf{side}((t, \mathsf{call}\, m(v')))$$

where, in the latter case, $(t, \mathsf{call}\, m(v'))$ is the corresponding call of $(t, \mathsf{ret}\, m(v))$. Thus, every move in h can be assigned a unique side polarity from $\{\mathcal{K}, \mathcal{L}\}$. For simplicity, we shall be tagging moves with a second index $Y \in \{\mathcal{K}, \mathcal{L}\}$ corresponding to their side polarity.

In this more restrictive nature of interaction, in which K and L' are separated, in addition to sequentiality in every thread we shall insist that a move made by the library in the $\mathcal{L}$ or $\mathcal{K}$ side must be followed by an O move from the *same* side.

Definition 12. We call a history $h \in \mathcal{H}_{\Psi,\Psi'}$ ***encapsulated*** if, for each thread t, we have that if

$$h = s_1\,(t,x)_{PY}\,s_2\,(t,x')_{OY'}\,s_3$$

and moves from t are absent from s_2 then $Y = Y'$. Moreover, if $L : \Psi \to \Psi'$, we set $\mathcal{H}^{\mathsf{enc}}_{\Psi,\Psi'} = \{h \in \mathcal{H}_{\Psi,\Psi'} \mid h \text{ encapsulated}\}$ and $[\![L]\!]_{\text{enc}} = [\![L]\!] \cap \mathcal{H}^{\mathsf{enc}}_{\Psi,\Psi'}$.

We define the corresponding linearisability notion as follows. First, let $\diamond \subseteq \mathcal{H}_{\Psi,\Psi'} \times \mathcal{H}_{\Psi,\Psi'}$ be the smallest binary relation on $\mathcal{H}_{\Psi,\Psi'}$ such that, for any X, X', and any $Y, Y' \in \{\mathcal{K}, \mathcal{L}\}$ with $Y \neq Y'$ and $t \neq t'$:

$$s_1(t,m)_{XY}(t',m')_{X'Y'}s_2 \;\;\diamond\;\; s_1(t',m')_{X'Y'}(t,m)_{XY}s_2$$

Definition 13 (Encapsulated linearisability). Given $h_1, h_2 \in \mathcal{H}^{\mathsf{enc}}_{\Psi,\Psi'}$, we say that h_1 is ***enc-linearised*** by h_2, and write $h_1 \vartriangleleft_{\text{enc}} h_2$, if $h_1(\vartriangleleft_{PO} \cup \diamond)^* h_2$ and h_2 is sequential.

A library $L : \Psi \to \Psi'$ can be *enc-linearised to* A, written $L \vartriangleleft_{\text{enc}} A$, if $A \subseteq \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'} \cap \mathcal{H}^{\mathsf{enc}}_{\Psi,\Psi'}$ and for any $h \in [\![L]\!]_{\text{enc}}$ there exists $h' \in A$ such that $h \vartriangleleft_{\text{enc}} h'$. We write $L \vartriangleleft_{\text{enc}} L'$ if $L \vartriangleleft_{\text{enc}} [\![L']\!]_{\text{enc}} \cap \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$.

Remark 14. Suppose $\Psi = \{m : \mathsf{int} \to \mathsf{int}\}$ and $\Psi' = \{m' : \mathsf{int} \to \mathsf{int}\}$. Histories from $\mathcal{H}_{\Psi,\Psi'}$ may contain the following actions only: $\mathsf{call}\, m'(i)_{O\mathcal{K}}$, $\mathsf{ret}\, m(i)_{O\mathcal{L}}$, $\mathsf{call}\, m(i)_{P\mathcal{L}}$, $\mathsf{ret}\, m'(i)_{P\mathcal{K}}$. Then $(\vartriangleleft_{PO} \cup \diamond)^*$ prevents $\mathsf{call}\, m(i)_{P\mathcal{L}}$ from being swapped with $\mathsf{ret}\, m(i)_{O\mathcal{L}}$ and, similarly, for $\mathsf{ret}\, m'(i)_{P\mathcal{K}}$ and $\mathsf{call}\, m'(i)_{O\mathcal{K}}$, i.e. it coincides with Definition 3 of [3].

Remark 15. The encapsulated framework implies that the client and the parameter library are independent entities. Consequently, whenever their interaction with the library involves two adjacent moves $(t,m)_{XY}(t',m')_{X'Y'}$ with $t \neq t', X \neq X'$, permuting them will also generate a valid interaction. This justifies the extra freedom in rearranging moves in Definition 13. The soundness of this intuition is validated in Lemma 39 and Theorem 40.

Example 16 (Parameterised multiset). We revisit the multiset library of Example 1 and extend it with a public method *reset*, which performs multiplicity resets to default values using an abstract method *default* as the default-value function (again, we use absolute values to avoid negative multiplicities). The extended library is shown in the RHS of Figure 2 and written $L_{\mathsf{mset2}} : \Psi \to \Psi'$, with $\Psi = \{\textit{default}\}$ and $\Psi' = \{\textit{count, update, reset}\}$. In contrast to the *update* method of L_{mset}, *reset* is not optimistic: it retrieves the lock upon its call, and only releases it before return. In particular, the method calls *default* while it retains the lock.

Observe that, were *default* able to externally call *update*, we would reach a deadlock: *default* would be keeping the lock while waiting for the return of a method that requires the lock. On the other hand, if the library is encapsulated then the latter scenario is not possible. In such a case, L_{mset2} linearises to the specification A_{mset2}, defined next. Let $A_{\mathsf{mset2}} = \{\, \pi_1(s) \mid s \in A^{\circ}_{\mathsf{mset2}} \,\}$ where:

$$A^{\circ}_{\mathsf{mset2}} = \{\, s \in \mathcal{H}^{\circ}_{\Psi,\Psi'} \mid \pi_1(s) \in \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'} \cap \mathcal{H}^{\mathsf{enc}}_{\Psi,\Psi'} \wedge \forall t.\, s \upharpoonright t \in \mathcal{S} \wedge s = (\_, I)_O\, s' \implies \forall i. I(i) = 0 \\ \wedge\; \forall s'(\_, I)_P(\_, J)_O \sqsubseteq_{\text{pre}} s.\; I = J \,\}$$

```
public run;  ...;
Lock lock;
struct {fun, arg, wait, retv} requests[N];

run = λ (f,x).
  requests[tid].fun := f;
  requests[tid].arg := x;
  requests[tid].wait := 1;
  while (requests[tid].wait)
    if (lock.tryacquire()) (
      for (t=0; t<N; t++)
        if (requests[t].wait) (
            requests[t].retv :=
                  requests[t].fun (requests[t].arg);
            requests[t].wait := 0;
        ); lock.release() );
  requests[tid].retv;
```

Figure 4: Flat combination library L_{fc}.

and the set $\mathcal{S}$ is now given by the grammar of Example 5 extended with the rule:

$$\mathcal{S} \to\ (t, \mathsf{call}\ \mathit{reset}(i), I)_{O\mathcal{K}}\ (t, \mathsf{call}\ \mathit{default}(i), I)_{P\mathcal{L}}\ (t, \mathsf{ret}\ \mathit{default}(j), I)_{O\mathcal{L}}\ (t, \mathsf{ret}\ \mathit{reset}(|j|), I')_{P\mathcal{K}}\ \mathcal{S}$$

with $I' = I[i \mapsto |j|]$. Our framework makes it possible to confirm that L_{mset2} enc-linearises to A_{mset2}.

2.2.3. Relational linearisability

We finally extend general linearisability to cater for situations where the client and the parameter library adhere to closure constraints expressed by relations $\mathcal{R}$ on histories. Let Ψ, Ψ' be sets of abstract and public methods respectively. The closure relations we consider are closed under permutations of methods outside $\Psi \cup \Psi'$: if $h\,\mathcal{R}\,h'$ and π is a (type-preserving) permutation on $\mathsf{Meths} \setminus (\Psi \cup \Psi')$ then $\pi(h)\,\mathcal{R}\,\pi(h')$. The requirement represents the fact that, apart from the method names from a library interface, the other method names are arbitrary and can be freely permuted without any observable effect. Thus, $\mathcal{R}$ should not be distinguishing between such names.

Definition 17 (Relational linearisability). Let $\mathcal{R} \subseteq \mathcal{H}_{\Psi,\Psi'} \times \mathcal{H}_{\Psi,\Psi'}$ be closed under permutations of names in $\mathsf{Meths} \setminus (\Psi \cup \Psi')$. Given $h_1, h_2 \in \mathcal{H}_{\Psi,\Psi'}$, we say that h_1 is ***$\mathcal{R}$-linearised*** by h_2, and write $h_1 \triangleleft_{\mathcal{R}} h_2$, if $h_1 (\triangleleft_{PO} \cup \mathcal{R})^* h_2$ and h_2 is sequential. A library $L : \Psi \to \Psi'$ can be *$\mathcal{R}$-linearised to A*, written $L \triangleleft_{\mathcal{R}} A$, if $A \subseteq \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$ and for any $h \in [\![L]\!]$ there exists $h' \in A$ such that $h \triangleleft_{\mathcal{R}} h'$. We write $L \triangleleft_{\mathcal{R}} L'$ if $L \triangleleft_{\mathcal{R}} [\![L']\!] \cap \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$.

Example 18. We consider a higher-order variant of an example from [3] that motivates relational linearisability. Flat combining [9] is a synchronisation paradigm that advocates the use of a single thread holding a global lock to process requests of all other threads. To facilitate this, threads share an array to which they write the details of their requests and wait either until they acquire a lock or their request has been processed by another thread. Once a thread acquires a lock, it executes all requests stored in the array and the outcomes are written to the array for access by the requesting threads.

Let $\Psi' = \{run \in \mathsf{Meths}_{(\theta\to\theta')\times\theta,\theta'}\}$. The library $L_{\mathsf{fc}} : \varnothing \to \Psi'$ in Figure 4 is built following the flat combining approach and, on acquisition of the global lock, the winning thread acts as a combiner of all registered requests. Note that the requests will be attended to one after another (thus guaranteeing mutual exclusion) and only one lock acquisition will suffice to process one array of requests. Using our framework, one can show that L_{fc} can be $\mathcal{R}$-linearised to the specification given by the library L_{spec} defined by

$$run = \lambda\,(f,x).\ \ (lock\,.\,acquire\,();\ \ \mathsf{let}\ \ result\ = f(x)\ \ \mathsf{in}\ \ lock\,.\,release\,();\ \ result\,)$$

where each function call in L_{spec} is protected by a lock. Observe that we cannot hope for $L_{\mathsf{fc}} \triangleleft L_{\mathsf{spec}}$, because clients may call library methods with functional arguments that recognise thread identity. Consequently, we can relate the two libraries only if context behaviour is guaranteed to be independent of thread identifiers. This can be expressed through $\triangleleft_{\mathcal{R}}$, where $\mathcal{R} \subseteq \mathcal{H}_{\varnothing,\Psi'} \times \mathcal{H}_{\varnothing,\Psi'}$ is a relation capturing thread-blind client behaviour (see Subsection 3.2 for details).

3. Library syntax and semantics

We now look at the concrete syntax of libraries and clients. Libraries comprise collections of typed methods whose argument and result types adhere to the grammar: $\theta ::= \mathsf{unit} \mid \mathsf{int} \mid \theta \to \theta \mid \theta \times \theta$.

We shall use three disjoint enumerable sets of names, referred to as Vars, Meths and Refs, to name respectively variables, methods and references. x, f (and their decorated variants) will be used to range over Vars; m will range over Meths; and r over Refs. Methods and references are implicitly typed, i.e. $\mathsf{Meths} = \biguplus_{\theta,\theta'} \mathsf{Meths}_{\theta,\theta'}$ and $\mathsf{Refs} = \mathsf{Refs}_{\mathsf{int}} \uplus \biguplus_{\theta,\theta'} \mathsf{Refs}_{\theta,\theta'}$, where $\mathsf{Meths}_{\theta,\theta'}$ contains names for methods of type $\theta \to \theta'$, $\mathsf{Refs}_{\mathsf{int}}$ contains names of integer references and $\mathsf{Refs}_{\theta,\theta'}$ contains names for references to methods of type $\theta \to \theta'$. We write $\uplus$ for disjoint set union.

The syntax for libraries and clients is given in Figure 5. Each library L begins with a series of method declarations (public or abstract) followed by a block B containing method implementations ($m = \lambda x.M$) and reference initialisations ($r := i$ or $r := \lambda x.M$). The typing rules ensure that each public method is implemented within the block, in contrast to abstract methods. Clients are parallel compositions of closed terms.

Terms M specify the shape of allowable method bodies. $()$ is the skip command, i ranges over integers, tid is the current thread identifier and $\oplus$ represents standard arithmetic operations. Thanks to higher-order references, we can simulate divergence by $(!r)()$, where $r \in \mathsf{Refs}_{\mathsf{unit},\mathsf{unit}}$ is initialised with $\lambda x^{\mathsf{unit}}.(!r)()$. Similarly, $\mathsf{while}\ M\ N$ can be simulated by $(!r)()$ after $r := \lambda x^{\mathsf{unit}}.\mathsf{let}\ y = M\ \mathsf{in}\ (\mathsf{if}\ y\ \mathsf{then}\ (N;(!r)())\ \mathsf{else}\ ())$. We also use the standard derived syntax for sequential composition, i.e. $M;N$ stands for $\mathsf{let}\ x = M\ \mathsf{in}\ N$, where x does not occur in N. For each term M, we write $\mathsf{Meths}(M)$ for the set of method names occurring in M. We use the same notation for method names in blocks and libraries.

Remark 19. In Section 2 we used lock-related operations in our example libraries (*acquire*, *tryacquire*, *release*), on the understanding that they can be coded using shared memory. We assume that both *acquire* and *release* are blocking, while *tryacquire* is not. *tryacquire* makes an attempt to acquire the associated lock and returns 0 if the attempt was not successful or 1 otherwise. Similarly, the array of Example 18 in the sequel can be constructed using references.

$$
\begin{array}{llll}
\textit{Libraries} & L ::= B \mid \mathsf{abstract}\ m;\ L \mid \mathsf{public}\ m;\ L & \textit{Clients} & K ::= M \,\|\cdots\|\, M \\
\textit{Blocks} & B ::= \epsilon \mid m = \lambda x.M;\ B \mid r := \lambda x.M;\ B \mid r := i;\ B & \textit{Values} & v ::= () \mid i \mid m \mid \langle v, v\rangle \\
\textit{Terms} & M ::= () \mid i \mid \mathsf{tid} \mid x \mid m \mid M \oplus M \mid \langle M, M\rangle \mid \pi_1 M \mid \pi_2 M \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M & & \\
& \qquad \mid \lambda x^{\theta}.M \mid xM \mid mM \mid \mathsf{let}\ x = M\ \mathsf{in}\ M \mid r := M \mid !r & &
\end{array}
$$

$$
\frac{}{\Gamma \vdash () : \mathsf{unit}} \quad \frac{}{\Gamma \vdash i : \mathsf{int}} \quad \frac{}{\Gamma \vdash \mathsf{tid} : \mathsf{int}} \quad \frac{\Gamma(x) = \theta}{\Gamma \vdash x : \theta} \quad \frac{m \in \mathsf{Meths}_{\theta,\theta'}}{\Gamma \vdash m : \theta \to \theta'} \quad \frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash M_0, M_1 : \theta}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_0 : \theta}
$$

$$
\frac{\Gamma \vdash M : \theta_1 \times \theta_2}{\Gamma \vdash \pi_i\, M : \theta_i \ \ (i = 1,2)} \quad \frac{\Gamma \vdash M_i : \theta_i \ \ (i = 1,2)}{\Gamma \vdash \langle M_1, M_2\rangle : \theta_1 \times \theta_2} \quad \frac{\Gamma \vdash M_1, M_2 : \mathsf{int}}{\Gamma \vdash M_1 \oplus M_2 : \mathsf{int}} \quad \frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'}
$$

$$
\frac{\Gamma(x) = \theta \to \theta' \quad \Gamma \vdash M : \theta}{\Gamma \vdash xM : \theta'} \quad \frac{m \in \mathsf{Meths}_{\theta,\theta'} \quad \Gamma \vdash M : \theta}{\Gamma \vdash mM : \theta'} \quad \frac{\Gamma \vdash M : \theta \quad \Gamma, x : \theta \vdash N : \theta'}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : \theta'}
$$

$$
\frac{r \in \mathsf{Refs}_{\mathsf{int}} \quad \Gamma \vdash M : \mathsf{int}}{\Gamma \vdash r := M : \mathsf{unit}} \quad \frac{r \in \mathsf{Refs}_{\theta,\theta'} \quad \Gamma \vdash M : \theta \to \theta'}{\Gamma \vdash r := M : \mathsf{unit}} \quad \frac{r \in \mathsf{Refs}_{\mathsf{int}}}{\Gamma \vdash !r : \mathsf{int}} \quad \frac{r \in \mathsf{Refs}_{\theta,\theta'}}{\Gamma \vdash !r : \theta \to \theta'}
$$

$$
\frac{}{\vdash_{\mathsf{B}} \epsilon : \varnothing} \quad \frac{m \in \mathsf{Meths}_{\theta,\theta'} \quad x : \theta \vdash M : \theta' \quad \vdash_{\mathsf{B}} B : \Psi}{\vdash_{\mathsf{B}} m = \lambda x.M;\ B : \Psi \uplus \{m\}} \quad \frac{r \in \mathsf{Refs}_{\theta,\theta'} \quad x : \theta \vdash M : \theta' \quad \vdash_{\mathsf{B}} B : \Psi}{\vdash_{\mathsf{B}} r := \lambda x.M;\ B : \Psi}
$$

$$
\frac{r \in \mathsf{Refs}_{\mathsf{int}} \quad \vdash_{\mathsf{B}} B : \Psi}{\vdash_{\mathsf{B}} r := i;\ B : \Psi} \quad \frac{\vdash_{\mathsf{B}} B : \Psi}{\mathsf{Meths}(B) \vdash_{\mathsf{L}} B : \varnothing \to \Psi} \quad \frac{\Psi \uplus \{m\} \vdash_{\mathsf{L}} L : \Psi' \to \Psi'' \quad m \in \Psi''}{\Psi \vdash_{\mathsf{L}} \mathsf{public}\ m; L : \Psi' \to \Psi''}
$$

$$
\frac{\Psi \uplus \{m\} \vdash_{\mathsf{L}} L : \Psi' \to \Psi'' \quad m \notin \Psi''}{\Psi \vdash_{\mathsf{L}} \mathsf{abstract}\ m; L : \Psi' \uplus \{m\} \to \Psi''} \quad \frac{\vdash M_j : \mathsf{unit} \ \ (j = 1, \cdots, N)}{\Psi \vdash_{\mathsf{K}} M_1 \|\cdots\| M_N : \mathsf{unit}}\ \forall j.\ \mathsf{Meths}(M_j) \subseteq \Psi
$$

Figure 5: Library syntax, and typing rules for terms ($\vdash$), blocks ($\vdash_{\mathsf{B}}$), libraries ($\vdash_{\mathsf{L}}$), clients ($\vdash_{\mathsf{K}}$).

For simplicity, we do not include private methods, yet the same effect could be achieved by storing them in higher-order references. As we explain in the next section, references present in library definitions are de facto private to the library. Note also that, according to our definition, sets of abstract and public methods are disjoint. However, given $m, m' \in \mathsf{Refs}_{\theta,\theta'}$, one can define a "public abstract" method with: $\mathsf{public}\ m;\ \mathsf{abstract}\ m';\ m = \lambda x^{\theta}.m'x$.

Terms are typed in environments $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$. Method blocks are typed through judgements $\vdash_{\mathsf{B}} B : \Psi$, where $\Psi \subseteq \mathsf{Meths}$. The judgments collect the names of methods defined in a block as well as making sure that the definitions respect types and are not duplicated. Also, the initialisation statements must comply with types.

Finally, we type libraries using statements of the form $\Psi \vdash_{\mathsf{L}} L : \Psi' \to \Psi''$, where $\Psi, \Psi', \Psi'' \subseteq \mathsf{Meths}$ and $\Psi' \cap \Psi'' = \varnothing$. The judgment $\varnothing \vdash_{\mathsf{L}} L : \Psi' \to \Psi''$ guarantees that any method occurring in L is present either in Ψ' or Ψ'', that all methods in Ψ' are declared as abstract and unimplemented, while all methods in Ψ'' are declared as public and defined. Thus, $\varnothing \vdash_{\mathsf{L}} L : \Psi \to \Psi'$ is a library in which Ψ, Ψ' are the abstract and public methods respectively. In this case, we also write $L : \Psi \to \Psi'$.

3.1. Semantics

The semantics of our system is given in several stages. First, we define an operational semantics for sequential and concurrent terms that may draw methods from a repository. We then adapt it to capture interactions of concurrent clients with closed libraries (no abstract methods). This notion is then used to define contextual approximation for

$$(L) \longrightarrow_{\text{lib}} (L, \varnothing, S_{\text{init}}) \qquad (r := i; B, \mathcal{R}, S) \longrightarrow_{\text{lib}} (B, \mathcal{R}, S[r \mapsto i])$$
$$(\mathsf{abstract}\ m; L, \mathcal{R}, S) \longrightarrow_{\text{lib}} (L, \mathcal{R}, S) \qquad (m = \lambda x.M; B, \mathcal{R}, S) \longrightarrow_{\text{lib}} (B, \mathcal{R}_{**}, S)$$
$$(\mathsf{public}\ m; L, \mathcal{R}, S) \longrightarrow_{\text{lib}} (L, \mathcal{R}, S) \qquad (r := \lambda x.M; B, \mathcal{R}, S) \longrightarrow_{\text{lib}} (B, \mathcal{R}_{**}, S[r \mapsto m])$$

$$(E[\mathsf{tid}], \mathcal{R}, S) \to_t (E[t], \mathcal{R}, S) \qquad (E[\mathsf{if}\ i_*\ \mathsf{then}\ M_1\ \mathsf{else}\ M_0], \mathcal{R}, S) \to_t (E[M_{j_*}], \mathcal{R}, S)$$
$$(E[i_1 \oplus i_2], \mathcal{R}, S) \to_t (E[i_{**}], \mathcal{R}, S) \qquad (E[\pi_j \langle v_1, v_2\rangle], \mathcal{R}, S) \to_t (E[v_j], \mathcal{R}, S)$$
$$(E[!r], \mathcal{R}, S) \to_t (E[S(r)], \mathcal{R}, S) \qquad (E[\mathsf{let}\ x = v\ \mathsf{in}\ M], \mathcal{R}, S) \to_t (E[M\{v/x\}], \mathcal{R}, S)$$
$$(E[r := i], \mathcal{R}, S) \to_t (E[()], \mathcal{R}, S[r \mapsto i]) \qquad (E[r := \lambda x.M], \mathcal{R}, S) \to_t (E[()], \mathcal{R}_{**}, S[r \mapsto m])$$
$$(E[\lambda x.M], \mathcal{R}, S) \to_t (E[m], \mathcal{R}_{**}, S) \qquad (E[mv], \mathcal{R}_*, S) \to_t (E[M\{v/x\}], \mathcal{R}_*, S)$$

$$E ::= \bullet \mid E \oplus M \mid i \oplus E \mid \mathsf{if}\ E\ \mathsf{then}\ M\ \mathsf{else}\ M \mid \pi_j\, E \mid \langle E, M\rangle \mid \langle v, E\rangle \mid mE \mid \mathsf{let}\ x = E\ \mathsf{in}\ M \mid r := E$$

$$\frac{(M, \mathcal{R}, S) \to_t (M', \mathcal{R}', S')}{(M_1 \| \cdots \| M_{t-1} \| M \| M_{t+1} \| \cdots \| M_N, \mathcal{R}, S) \Longrightarrow (M_1 \| \cdots \| M_{t-1} \| M' \| M_{t+1} \| \cdots \| M_N, \mathcal{R}', S')}\ (K_N)$$

Figure 6: . Evaluation rules for libraries ($\longrightarrow_{\text{lib}}$), terms ($\to_t$) and clients ($\Longrightarrow$). In the rules above we use the conditions/notation: $\mathcal{R}_{**} = \mathcal{R} \uplus (m \mapsto \lambda x.M)$, $i_{**} = i_1 \oplus i_2$, $\mathcal{R}_*(m) = \lambda x.M$, and $j_* = 0$ iff $i_* = 0$.

arbitrary libraries. Finally, we introduce a trace semantics of arbitrary libraries, which generates the histories on which our notions of linearisability are based.

3.1.1. Library-client evaluation

Libraries, terms and clients are evaluated in environments comprising:

- A method environment $\mathcal{R}$, called *own-method repository*, which is a finite partial map on Meths assigning to each m in its domain, with $m \in \mathsf{Meths}_{\theta,\theta'}$, a term of the form $\lambda y.M$ (we omit type-superscripts from bound variables for economy).
- A finite partial map $S : \mathsf{Refs} \rightharpoonup (\mathbb{Z} \cup \mathsf{Meths})$, called *store*, which assigns to each r in its domain an integer (if $r \in \mathsf{Refs}_{\mathsf{int}}$) or name from $\mathsf{Meths}_{\theta,\theta'}$ (if $r \in \mathsf{Refs}_{\theta,\theta'}$).

The evaluation rules are presented in Figure 6, where we also define *evaluation contexts* E.

Remark 20. We shall assume that reference names used in libraries are library-private, i.e. sets of reference names used in different libraries are assumed to be disjoint. Similarly, when libraries are being used by client code, this is done on the understanding that the references available to that code do not overlap with those used by libraries. Still, for simplicity, we shall rely on a single set Refs of references in our operational rules.

First we evaluate the library to create an initial repository and store. This is achieved by the first set of rules in Figure 6, where we assume that S_{init} is empty. Thus, library evaluation produces a tuple $(\epsilon, \mathcal{R}_0, S_0)$ including a method repository and a store, which can be used as the initial repository and store for evaluating $M_1 \| \cdots \| M_N$ using the (K_N) rule. We shall call the latter evaluation semantics for clients (denoted by $\Longrightarrow$) the *multi-threaded operational semantics*. The latter relies on closed-term reduction ($\to_t$), whose rules are given in the middle group, where t is the current thread index. Note that the rules for $E[\lambda x.M]$ in the middle group, along with those for $m = \lambda x.M$ and $r := \lambda x.M$ in the first group, involve the creation of a fresh method name m, which is used to put the function in the repository $\mathcal{R}$. Name creation is non-deterministic: any fresh m of the appropriate type can be chosen.

We define termination for clients linked with libraries that have no abstract methods.

Recall our convention (Remark 20) that L and $M_1, \cdots, M_N$ must access disjoint parts of the store. Terms $M_1, \cdots, M_N$ can share reference names, though.

Definition 21. Let $L : \varnothing \to \Psi'$ and $\Psi' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$. We say that $M_1 \| \cdots \| M_N$ *terminates with linked library* L if $(M_1 \| \cdots \| M_N, \mathcal{R}_0, S_0) \Longrightarrow^* (() \| \cdots \| (), \mathcal{R}, S)$, for some $\mathcal{R}, S$, where $(L) \longrightarrow^*_{\text{lib}} (\epsilon, \mathcal{R}_0, S_0)$. We then write $\mathsf{link}\ L\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$.

We shall build a notion of contextual approximation of libraries on top of termination: one library will be said to approximate another if, whenever the former terminates when composed with any parameter library and client, so does the latter.

We will be considering the following notions for composing libraries. Let us denote a library L as $L = D; B$, where D contains all the (public/abstract) method declarations of L, and B is its method block. We write $\mathsf{Refs}(L)$ for the set of references in L. Let $L_1 : \Psi_1 \to \Psi_2$ be of the form $D_1; B_1$. Given $L_2 : \Psi_1' \to \Psi_2'$ $(= D_2; B_2)$ such that $\Psi_2 \cap \Psi_2' = \mathsf{Refs}(L_1) \cap \mathsf{Refs}(L_2) = \varnothing$, $\Psi = \{m_1, \cdots, m_n\} \subseteq \Psi_2$ and $L' : \varnothing \to \Psi_1, \Psi'$, we define the *union* of L_1 and L_2, the Ψ*-hiding* of L_1, and the *sequencing* of L' with L_1 respectively as:

$$\begin{aligned}
L_1 \cup L_2 : (\Psi_1 \cup \Psi_1') \smallsetminus (\Psi_2 \cup \Psi_2') \to \Psi_2 \cup \Psi_2' \quad &= (D_1; B_1) \cup (D_2; B_2) = D_1'; D_2'; B_1; B_2 \\
L_1 \smallsetminus \Psi : \Psi_1 \to (\Psi_2 \smallsetminus \Psi) \quad &= (D_1; B_1) \smallsetminus \Psi = D_1''; B_1'\{!r_1/m_1\} \cdots \{!r_n/m_n\} \\
L' \mathbin{\boldsymbol{;}} L_1 : \varnothing \to \Psi_2, \Psi' \quad &= (L' \cup L_1) \smallsetminus \Psi_1
\end{aligned}$$

where D_1' is D_1 with any $\mathsf{abstract}\, m$ declaration removed for $m \in \Psi_2'$, dually for D_2'; and where D_1'' is D_1 without $\mathsf{public}\ m$ declarations for $m \in \Psi$ and each r_i is a fresh reference matching the type of m_i, and B_1' is obtained from B_1 by replacing each $m_i = \lambda x.M$ by $r_i \coloneqq \lambda x.M$. Thus, the union of libraries L_1 and L_2 corresponds to merging their code and removing any $\mathsf{abstract}$ declarations for methods defined in the union. On the other hand, the hiding of a public method simply renders it private via the use of references. Sequencing allows for the following notion.

Definition 22. Given $L_1, L_2 : \Psi \to \Psi'$, we say that L_1 ***contextually approximates*** L_2, written $L_1 \sqsubseteq L_2$, if for all $L' : \varnothing \to \Psi, \Psi''$ and $\Psi', \Psi'' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$, if $\mathsf{link}\ L' \mathbin{\boldsymbol{;}} L_1\ \mathsf{in} (M_1 \| \cdots \| M_N) \Downarrow$ then $\mathsf{link}\ L' \mathbin{\boldsymbol{;}} L_2\ \mathsf{in} (M_1 \| \cdots \| M_N) \Downarrow$. In this case, we also say that L_2 ***contextually refines*** L_1.

Note that, according to this definition, the parameter library L' may communicate directly with the client terms through a common interface Ψ''. We shall refer to this case as the *general* case. Later on, we shall also consider more restrictive testing scenarios in which this possibility of explicit communication is removed. Moreover, from the disjointness conditions in the definitions of sequencing and linking we have that L_i, L' and $M_1 \| \cdots \| M_N$ access pairwise disjoint parts of the store.

Remark 23. Our ultimate goal will be to show that our notion of linearisability, written $\triangleleft$, provides a sound method for proving contextual approximation/refinement, written $\sqsubseteq$. Recall that in order to establish $L_1 \triangleleft L_2$, one has to exhibit a subset A_2 of *sequential* histories taken from $[\![L_2]\!]$ such that L_1 is linearisable to A_2, written $L_1 \triangleleft A_2$.

3.1.2. Trace semantics

Building on the earlier semantics, we next introduce a trace semantics of libraries in the spirit of game semantics [14]. As mentioned in Section 2, the behaviour of a library will be represented as an exchange of moves between two players called P and O, representing the library and its corresponding context respectively. The context consists of the client of the library as well as the parameter library, with an index on each move ($\mathcal{K}/\mathcal{L}$) specifying which of them is involved in the move.

In contrast to the semantics of the previous section, we handle scenarios in which methods need not be present in the repository $\mathcal{R}$. Calls to such undefined methods are represented by labelled transitions – calls to the context made on behalf of the library (P). The calls can later be responded to with labelled transitions corresponding to returns, made by the context (O). On the other hand, O is able to invoke methods in $\mathcal{R}$, which will also be represented through suitable labels. Because we work in a higher-order setting, calls and returns made by both players may involve methods as arguments or results. Such methods also become available for future calls: function arguments/results supplied by P are added to the repository and can later be invoked by O, while function arguments/results provided by O can be queried in the same way as abstract methods.

The trace semantics utilises configurations that carry more components than the previous semantics. We define two kinds of configurations:

$$\textit{O-configurations } (\mathcal{E}, -, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \quad \text{and} \quad \textit{P-configurations } (\mathcal{E}, M, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$$

where the component $\mathcal{E}$ is an *evaluation stack*, that is, a stack of the form $[X_1, X_2, \cdots, X_n]$ with each X_i being either an evaluation context or a method name. On the other hand, $\mathcal{P} = (\mathcal{P}_\mathcal{L}, \mathcal{P}_\mathcal{K})$ with $\mathcal{P}_\mathcal{L}, \mathcal{P}_\mathcal{K} \subseteq \mathsf{dom}(\mathcal{R})$ being sets of *public* method names, and $\mathcal{A} = (\mathcal{A}_\mathcal{L}, \mathcal{A}_\mathcal{K})$ is a pair of sets of *abstract* method names. $\mathcal{P}$ will be used to record all the method names produced by P and passed to O: those passed to $O\mathcal{K}$ are stored in $\mathcal{P}_\mathcal{K}$, while those passed to $O\mathcal{L}$ are kept in $\mathcal{P}_\mathcal{L}$. Inside $\mathcal{A}$, the story is the opposite one: $\mathcal{A}_\mathcal{K}$ ($\mathcal{A}_\mathcal{L}$) stores the method names produced by $O\mathcal{K}$ (resp. $O\mathcal{L}$) and passed to P. Consequently, the sets of names stored in $\mathcal{P}_\mathcal{L}, \mathcal{P}_\mathcal{K}, \mathcal{A}_\mathcal{L}, \mathcal{A}_\mathcal{K}$ will always be disjoint.

Given a pair $\mathcal{P}$ as above and a set $Z \subseteq \mathsf{Meths}$, we write $\mathcal{P} \cup_\mathcal{K} Z$ for the pair $(\mathcal{P}_\mathcal{L}, \mathcal{P}_\mathcal{K} \cup Z)$. We define $\cup_\mathcal{L}$ in a similar manner, and extend it to pairs $\mathcal{A}$ as well. Moreover, given $\mathcal{P}$ and $\mathcal{A}$, we let $\phi(\mathcal{P}, \mathcal{A})$ be the set of *fresh* method names for $\mathcal{P}, \mathcal{A}$: $\phi(\mathcal{P}, \mathcal{A}) = \mathsf{Meths} \setminus (\mathcal{P}_\mathcal{L} \cup \mathcal{P}_\mathcal{K} \cup \mathcal{A}_\mathcal{L} \cup \mathcal{A}_\mathcal{K})$.

We give the rules generating the trace semantics in Figure 7. Note that the rules are parameterised by: P/O and Y, which together determine the polarity of the next move; C/R, which stands for the move being a call (*C*) or a return (*R*) respectively. The rules depict the intuition presented above. When in an O-configuration, the context may issue a call to a public method $m \in \mathcal{P}_Y$ and pass control to the library (rule (OCY)). Note that, when this occurs, the name m is added to the evaluation stack $\mathcal{E}$ and a P-configuration is obtained. From there on, the library will compute internally using rule (INT), until: it either needs to evaluate an abstract method (i.e. some $m' \in \mathcal{A}_Y$), and hence issues a call via rule (PCY); or it completes its computation and returns the call (rule (PRY)). Calls to abstract methods, on the other hand, are met either by further calls to public methods (via (OCY)), or by returns (via (ORY)).

Finally, we extend the trace semantics to a concurrent setting where a fixed number of N-many threads run in parallel. Each thread has separate evaluation stack and term components, which we write as $\mathcal{C} = (\mathcal{E}, X)$ (where X is a term or "–"). Thus, a configuration now is of the following form:

$$N\textit{-configuration} \;\; (\mathcal{C}_1 \| \cdots \| \mathcal{C}_N, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$$

where, for each i, $\mathcal{C}_i = (\mathcal{E}_i, X_i)$ and $(\mathcal{E}_i, X_i, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ is a sequential configuration. We shall abuse notation a little and write $(\mathcal{C}_i, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ for $(\mathcal{E}_i, X_i, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$. The

(INT) $(\mathcal{E}, M, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \rightarrow_t (\mathcal{E}, M', \mathcal{R}', \mathcal{P}, \mathcal{A}, S')$, given that $(M, \mathcal{R}, S) \rightarrow_t (M', \mathcal{R}', S')$ and $\mathsf{dom}(\mathcal{R}' \smallsetminus \mathcal{R})$ consists of names that do not occur in $\mathcal{E}, \mathcal{A}$.

(PCY) $(\mathcal{E}, E[mv], \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xrightarrow{\mathsf{call}\, m(v')_{PY}}_t (m :: E :: \mathcal{E}, -, \mathcal{R}', \mathcal{P}', \mathcal{A}, S)$, given $m \in \mathcal{A}_Y$ and (**P**).

(OCY) $(\mathcal{E}, -, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xrightarrow{\mathsf{call}\, m(v)_{OY}}_t (m :: \mathcal{E}, M\{v/x\}, \mathcal{R}, \mathcal{P}, \mathcal{A}', S)$, given $m \in \mathcal{P}_Y$, $\mathcal{R}(m) = \lambda x.M$ and (**O**).

(PRY) $(m :: \mathcal{E}, v, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xrightarrow{\mathsf{ret}\, m(v')_{PY}}_t (\mathcal{E}, -, \mathcal{R}', \mathcal{P}', \mathcal{A}, S)$, given $m \in \mathcal{P}_Y$ and (**P**).

(ORY) $(m :: E :: \mathcal{E}, -, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xrightarrow{\mathsf{ret}\, m(v)_{OY}}_t (\mathcal{E}, E[v], \mathcal{R}, \mathcal{P}, \mathcal{A}', S)$, given $m \in \mathcal{A}_Y$ and (**O**).

(**P**) If v contains the names $m_1, \cdots, m_k$ then $v' = v\{m'_i/m_i \mid 1 \le i \le k\}$ with each m'_i being a fresh name. Moreover, $\mathcal{R}' = \mathcal{R} \uplus \{m'_i \mapsto \lambda x.m_i x \mid 1 \le i \le k\}$ and $\mathcal{P}' = \mathcal{P} \cup_Y \{m'_1, \cdots, m'_k\}$.

(**O**) If v contains names $m_1, \cdots, m_k$ then $m_i \in \phi(\mathcal{P}, \mathcal{A})$, for each i, and $\mathcal{A}' = \mathcal{A} \cup_Y \{m_1, \cdots, m_k\}$.

Figure 7: Trace semantics rules. The rule (INT) is for embedding internal rules. In the rule (PCY), the library (P) calls one of its abstract methods (either the original ones or those acquired via interaction), while in (PRY) it returns from such a call. The rules (OCY) and (ORY) are dual and represent actions of the context. In all of the rules, whenever we write $m(v)$ or $m(v')$, we assume that the type of v matches the argument type of m.

concurrent traces are produced by the following two rules

$$\frac{(\mathcal{C}_i, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \rightarrow_i (\mathcal{C}', \mathcal{R}, \mathcal{P}, \mathcal{A}, S')}{(\mathcal{C}_1 \| \cdots \| \mathcal{C}_N, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \Longrightarrow (\mathcal{C}_1 \| \cdots \| \mathcal{C}_{i-1} \| \mathcal{C}' \| \mathcal{C}_{i+1} \| \cdots \| \mathcal{C}_N, \mathcal{R}, \mathcal{P}, \mathcal{A}, S')} \text{ (PINT)}$$

$$\frac{(\mathcal{C}_i, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xrightarrow{x_{XY}}_i (\mathcal{C}', \mathcal{R}, \mathcal{P}, \mathcal{A}, S')}{(\mathcal{C}_1 \| \cdots \| \mathcal{C}_N, \mathcal{R}, \mathcal{P}, \mathcal{A}, S) \xRightarrow{(i,x)_{XY}} (\mathcal{C}_1 \| \cdots \| \mathcal{C}_{i-1} \| \mathcal{C}' \| \mathcal{C}_{i+1} \| \cdots \| \mathcal{C}_N, \mathcal{R}, \mathcal{P}, \mathcal{A}, S')} \text{ (PEXT)}$$

with the proviso that the names freshly produced internally in (PINT) are fresh for the whole of $\vec{\mathcal{C}}$.

We can now define the trace semantics of a library L. We call a configuration component $\mathcal{C}_i$ ***final*** if it is in one of the following forms, for O- and P-configurations respectively: $\mathcal{C}_i = ([], -)$ or $\mathcal{C}_i = ([], ())$. We call $(\vec{\mathcal{C}}, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ final just if $\vec{\mathcal{C}} = \mathcal{C}_1 \| \cdots \| \mathcal{C}_N$ and each $\mathcal{C}_i$ is final.

Definition 24. For each $L : \Psi \rightarrow \Psi'$, we define the N-trace semantics of L to be:

$$[\![L]\!]_N = \{\, s \mid (\vec{\mathcal{C}}_0, \mathcal{R}_0, (\varnothing, \Psi'), (\Psi, \varnothing), S_0) \xRightarrow{s}{}^* \rho \wedge \rho \text{ final} \,\}$$

where $\vec{\mathcal{C}}_0 = ([], -) \| \cdots \| ([], -)$ and $(L) \longrightarrow^*_{\text{lib}} (\epsilon, \mathcal{R}_0, S_0)$.

For economy, in the sequel we might be dropping the index N from $[\![L]\!]_N$. We conclude the presentation of the trace semantics by providing a semantics for library contexts.

Recall that in our setting (Figure 1) a library $L : \Psi \to \Psi'$ is deployed in a context consisting of a parameter library $L' : \varnothing \to \Psi, \Psi''$ and a concurrent composition of client threads $\Psi', \Psi'' \vdash M_i : \mathsf{unit}$ ($i = 1, \cdots, N$). We shall write $\mathsf{link}\ L'\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)$, or simply C, to refer to such contexts.

Definition 25. Let $\Psi', \Psi'' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$ and $L' : \varnothing \to \Psi, \Psi''$. We define the semantics of the context formed by L' and $M_1, \cdots, M_N$ to be:

$$[\![\mathsf{link}\ L'\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!] = \{\, s \mid (\vec{\mathcal{C}}_0, \mathcal{R}_0, (\Psi, \varnothing), (\varnothing, \Psi'), S_0) \xRightarrow{s}{}^{*} \rho \wedge \rho \text{ final} \,\}$$

where $(L') \longrightarrow^{*}_{\text{lib}} (\epsilon, \mathcal{R}_0, S_0)$ and $\vec{\mathcal{C}}_0 = ([\,], M_1) \| \cdots \| ([\,], M_N)$.

Lemma 26. *For any $L : \Psi \to \Psi'$, $L' : \varnothing \to \Psi, \Psi''$ and $\Psi', \Psi'' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$ we have $[\![L]\!]_N \subseteq \mathcal{H}_{\Psi,\Psi'}$ and $[\![\mathsf{link}\ L'\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!] \subseteq \mathcal{H}^{co}_{\Psi,\Psi'}$.*

3.2. *Proofs of examples*

With the definition of $[\![L]\!]$ in place, we can finally revisit the linearisability claims anticipated in Examples 1, 16 and 18.

Recall the multiset library L_{mset} and the specification A_{mset} of Example 1 and Figure 2. We show that $L_{\mathsf{mset}} \lhd A_{\mathsf{mset}}$. More precisely, taking an arbitrary history $h \in [\![L_{\mathsf{mset}}]\!]$ we show that h can be rearranged using $\lhd^{*}_{PO}$ to match an element of A_{mset}. We achieve this by identifying, for each O-move $(t, x)_O$ and its following P-move $(t, x')_P$ in h, a *linearisation point* between them, i.e. a place in h to which $(t, x)_O$ can moved right and to which $(t, x')_P$ can be moved left so that they become consecutive and, moreover, the resulting history is still produced by L_{mset}. After all these rearrangements, we obtain a sequential history $\hat{h}$ such that $h \lhd \hat{h}$ and $\hat{h}$ is also produced by L_{mset}. It then suffices to show that $\hat{h} \in A_{\mathsf{mset}}$.

Lemma 27 (Multiset). *L_{mset} linearises to A_{mset}.*

Proof. Given some $h \in [\![L_{\mathsf{mset}}]\!]$, let us assume that h has been generated by a sequence $\rho_1 \Rightarrow \rho_2 \Rightarrow \cdots \Rightarrow \rho_k$ of atomic transitions and that the variable F of L_{mset} is instantiated with a reference r_F. We demonstrate the linearisation points for pairs of (O, P) moves in h, by case analysis on the moves (we drop $\mathcal{K}$ indices from moves as they are ubiquitous). Line numbers below refer to the LHS of Figure 2.

1. $h = \cdots (t, \mathsf{call}\ \mathit{cnt}(i))_O\, s\, (t, \mathsf{ret}\ \mathit{cnt}(i'))_P \cdots$. Here the linearisation point (LP) is the configuration ρ_j that dereferences r_F as per line 5 in L_{mset} (the $!F$ expression).
2. $h = \cdots (t, \mathsf{call}\ \mathit{upd}(i, m))_O\, s\, (t, \mathsf{call}\ m(j))_P \cdots$. The LP is the dereferencing of r_F in line 5 (called from within *update*).
3. $h = \cdots (t, \mathsf{ret}\ m(j'))_O\, s\, (t, \mathsf{ret}\ \mathit{upd}(|j'|))_P \cdots$. The LP is the update of r_F in line 13.
4. $h = \cdots (t, \mathsf{ret}\ m(j'))_O\, s\, (t, \mathsf{call}\ m(j''))_P \cdots$. The LP is the dereferencing of r_F in line 11.

Each of the linearisation points above specifies a PO-rearrangement of moves. For instance, for $h = s_0\, (t, \mathsf{call}\ \mathit{cnt}(i))_O\, s\, (t, \mathsf{ret}\ \mathit{cnt}(i'))_P\, s'$, let $s = s_1 s_2$ where $s_0\, (t, \mathsf{call}\ \mathit{cnt}(i))_O\, s_1$ is the prefix of h produced by $\rho_1 \Rightarrow \rho_2 \Rightarrow \cdots \Rightarrow \rho_j$. The rearrangement of h is then $\hat{h} = s_0\, s_1\, (t, \mathsf{call}\ \mathit{cnt}(i))_O\, (t, \mathsf{ret}\ \mathit{cnt}(i'))_P\, s_2\, s'$. We thus obtain $h \lhd^{*}_{PO} \hat{h}$.

The selection of linearisation points is such that it guarantees that $\hat{h} \in [\![L_{\mathsf{mset}}]\!]$. E.g. in case 1, the transitions occurring in thread t between $(t, \mathsf{call}\ \mathit{cnt}(i))_O$ and configuration ρ_j do not access r_F. Hence, we can postpone them and fire them in sequence just before ρ_j. After ρ_{j+1} and until $(t, \mathsf{ret}\ \mathit{cnt}(i'))_P$ there is again no access of r_F in t and we can thus bring forward the corresponding transitions just after ρ_{j+1}. Similar reasoning applies to case 2. In case 4, we reason similarly but also take into account that rendering the acquisition of the lock by t atomic is sound (i.e. the semantics can produce the rearranged history). Case 3 is similar, but we also use the fact that the access to r_F in lines 10-15 is inside the lock, and hence postponing dereferencing (line 11) to occur in sequence before update (line 13) is sound.

Now, any transition sequence α which produces $\hat{h}$ (in $[\![L_{\mathsf{mult}}]\!]$) can be used to derive an annotated history $h^\circ \in A^\circ_{\mathsf{mult}}$, by attaching to each move in $\hat{h}$ the multiset represented in the configuration that produces the move (ρ produces the move x if $\rho \xRightarrow{x} \rho'$ in α). By projection we obtain $\hat{h} \in A_{\mathsf{mult}}$. ☐

Lemma 28 (Parameterised multiset). *L_{mset2} enc-linearises to A_{mset2}.*

Proof. Again, we identify linearisation points, this time for given $h \in [\![L_{\mathsf{mult2}}]\!]_{\mathsf{enc}}$. For cases 1-4 as above we reason as in Lemma 27. For *reset* we have the following case.

$$h = s\,(t, \mathsf{call}\ \mathit{reset}(i))_{O\mathcal{K}}\, s_1(t, \mathsf{call}\ \mathit{default}(j))_{P\mathcal{L}}\, s_2(t, \mathsf{ret}\ \mathit{default}(j'))_{O\mathcal{L}}\, s_3(t, \mathsf{ret}\ \mathit{reset}(|j'|))_{P\mathcal{K}} \cdots$$

Here, we need a linearisation point for all four moves above. We pick this to be the point corresponding to the update of the multiset reference F on lines 24-25 (Figure 2, RHS). We now transform h to $\hat{h}$ so that the four moves become consecutive, in two steps:

- Let us write s_3 as $s_3 = s_3^1 s_3^2$, where the split is at the linearisation point. Since the lock is constantly held by thread t in $s_2 s_3^1$, there can be no calls or returns to *default* in $s_2 s_3^1$. Hence, all moves in $s_2 s_3^1$ are in component $\mathcal{K}$ and can be transposed with the $\mathcal{L}$-moves above, using $\diamond^*$, to obtain $h' = s\,(t, \mathsf{call}\ \mathit{reset}(i))_{O\mathcal{K}}\, s_1\, s_2\, s_3^1(t, \mathsf{call}\ \mathit{default}(j))_{P\mathcal{L}}$ $(t, \mathsf{ret}\ \mathit{default}(j'))_{O\mathcal{L}}\, s_3^2(t, \mathsf{ret}\ \mathit{reset}(|j'|))_{P\mathcal{K}} \cdots$
- Next, by PO-rearrangement we obtain $\hat{h} = s\, s_1\, s_2\, s_3^1(t, \mathsf{call}\ \mathit{reset}(i))_{O\mathcal{K}}(t, \mathsf{call}\ \mathit{default}(j))_{P\mathcal{L}}$ $(t, \mathsf{ret}\ \mathit{default}(j'))_{O\mathcal{L}}(t, \mathsf{ret}\ \mathit{reset}(|j'|))_{P\mathcal{K}}\, s_3^2 \cdots$. Thus, $h(\lhd_{PO} \cup \diamond)^* \hat{h}$.

To prove that $\hat{h} \in A_{\mathsf{mult2}}$ we work as in Lemma 27, i.e. via showing that $\hat{h} \in [\![L_{\mathsf{mult2}}]\!]_{\mathsf{enc}}$. For the latter, we rely on the fact that the linearisation point was taken at the reference update point (so that any dereferencings from other threads are preserved), and that the dereferences of lines 22 and 23 are within the same lock as the update. ☐

For our last example, recall the flat combination library $L_{\mathsf{fc}} : \varnothing \to \Psi'$ of Example 18, and Figure 4, along with its specification library $L_{\mathsf{spec}} : \varnothing \to \Psi'$, where $\Psi' = \{\mathit{run} \in \mathsf{Meths}_{(\theta \to \theta') \times \theta, \theta'}\}$.

Remark 29. It is worth observing that in the higher-order setting a client thread may try to call *run*, even though the previous call to *run* by the same thread did not complete yet. This scenario happens, for example, when the first call to *run* passes a functional argument to the library that itself calls *run*. Observe that in this case both L_{fc} and L_{spec} will deadlock. Consequently, non-trivial histories (all calls are matched by returns) arise only if each client thread uses run serially, i.e. without nesting.

Let $\mathcal{R} = <^*$, where $< \subseteq \mathcal{H}_{\varnothing,\Psi'} \times \mathcal{H}_{\varnothing,\Psi'}$ is the smallest relation such that (for economy we omit methods from calls/returns):

- $s_1(t, \mathsf{call}\ )_P s_2(t, \mathsf{ret}\ )_O s_3 < s_1(t', \mathsf{call}\ )_P s_2(t', \mathsf{ret}\ )_O s_3$
- $s_1(t, \mathsf{call}\ )_P s_2(t, \mathsf{call}\ )_O s_3(t, \mathsf{ret}\ )_P s_4(t, \mathsf{ret}\ )_O s_5 < s_1(t', \mathsf{call}\ )_P s_2(t', \mathsf{call}\ )_O s_3(t', \mathsf{ret}\ )_P s_4(t', \mathsf{ret}\ )_P s_5$

for any s_1, s_2, s_3, s_4, s_5 such that s_2, s_4 do not contain any t-moves.

Intuitively, $<$ is about piecewise delegation of client computations to other existing threads subject to forming a correct history. Because the results do not change, this condition corresponds to thread-blind client behaviour.

Lemma 30 (Flat combining). *L_{fc} $\mathcal{R}$-linearises to L_{spec}.*

Proof. Observe that histories from $[\![L_{\mathsf{spec}}]\!]$ feature threads built from segments of one of the three forms:

- $(t, \mathsf{call}\,\mathsf{run}(f,x))_O\ (t, \mathsf{call}\, f(x'))_P \cdots (t, \mathsf{ret}\, f(v))_O\ (t, \mathsf{ret}\,\mathsf{run}(v')))_P$, or
- $(t', \mathsf{call}\, w(v))_O\ (t', \mathsf{call}\, w'(v'))_P$, where w is a name introduced in an earlier move $(t'', x)_P$ and w' is a corresponding name introduced by the move preceding $(t'', x)_P$ in t'', or
- $(t', \mathsf{ret}\, w'(v''))_O\ (t', \mathsf{ret}\, w(v'''))_P$ such that a segment $(t', \mathsf{call}\, w(v))_O\ (t', \mathsf{call}\, w'(v'))_P$ already occurred earlier.

The first shape represents interaction of the client with the library: a call to run followed by a call to f, possibly some intermediate computation (using calls/returns to higher-order values that have been introduced in the trace), and a return of f followed by a return of run. The value introduced in the last return may well be a function, which – along with method names introduced earlier – provides method names that can be used in calls and returns later. As these methods are related to concrete functions, our trace semantics interprets them in a symbolic manner: each call is forwarded to the move preceding the one in which it was introduced. Note that threads can exchange higher-order values, so we need to allow for scenarios in which the three kinds of interaction are located in different threads.

We shall refer to moves in the second and third kind of segments as *inspection moves* and write ϕ to refer to sequences built exclusively from such sequences. Note that $\cdots$ in the first kind of block also stand for a segment of inspection moves in t.

Let us write $\mathcal{X}$ for the subset of $[\![L_{\mathsf{spec}}]\!]$ containing (sequential) plays of the form:

$$\begin{aligned}&(t_0, \mathsf{call}\,\mathsf{run}(f_0,x_0)(t_0, \mathsf{call}\, f_0(x_0')\phi_0(t_0, \mathsf{ret}\, f_0(v_0))(t_0, \mathsf{ret}\,\mathsf{run}(v_0'))\,\phi_1\\ &(t_1, \mathsf{call}\,\mathsf{run}(f_1,x_1)(t_1, \mathsf{call}\, f_1(x_1')\phi_2(t_1, \mathsf{ret}\, f_1(v_1))(t_1, \mathsf{ret}\,\mathsf{run}(v_1'))\,\phi_3\\ &\cdots(t_k, \mathsf{call}\,\mathsf{run}(f_k,x_k)(t_k, \mathsf{call}\, f_k(x_k')\phi_{2k}(t_k, \mathsf{ret}\, f_k(v_k))(t_k, \mathsf{ret}\,\mathsf{run}(v_k'))\,\phi_{2k+1}.\end{aligned}$$

where ϕ_{2j}, ϕ_{2j+1} may also contain inspection moves not in t_j. We take $\mathcal{X}$ to be our linearisation target (specification).

Consider $h_1 \in [\![L_{\mathsf{fc}}]\!]$. Threads in h_1 are built from blocks of shapes:

$$\begin{aligned}&(t, \mathsf{call}\,\mathsf{run}(f,x)_O\ ((t, \mathsf{call}\, f_j(x_j')_P\ \phi_j(t, \mathsf{ret}\, f_j(v_j)_O))^*\ (t, \mathsf{ret}\,\mathsf{run}(v'))_P\\ &\text{or}\ \ (t', \mathsf{call}\, w(v))_O\,(t', \mathsf{call}\, w'(v'))_P\ \ \text{or}\ \ (t', \mathsf{ret}\, w'(v''))_O\,(t', \mathsf{ret}\, w(v'''))_P.\end{aligned}$$

In the first case, the j's are meant to represent possibly different values used in each iteration. In the second kind of block, w needs to be introduced earlier by some $(t'', x)_P$ move and w' is then a name introduced by the preceding move. For the third kind, an earlier calling sequence of the second kind must exist in the same thread.

Observe that each segment $S_j = (t, \mathsf{call}\, f_j(x_j')_P\ \phi_j(t, \mathsf{ret}\, f_j(v_j))_O$ in t must be preceded (in h_1) by a matching public call $(t', \mathsf{call}\,\mathsf{run}(f_j, x_j))_O$ followed by a corresponding return $(t', \mathsf{ret}\,\mathsf{run}(v_j))_P$, where t' need not be equal to t. We can obtain the requisite h (for $\lhd_{\mathcal{R}}$) by changing t to t' in the whole of S_j for each S_j. Note that run-moves are not affected and we get $h_1\,\mathcal{R}^*\,h$.

Note that, due to locking and sequentiality of loops, the segments S_j must be disjoint in h_1, although they may be interleaved with inspection moves from other threads. We shall show how to obtain $h_2 \in \mathcal{X}$ with $h \lhd_{PO}^* h_2$.

- First the call to run associated with each S_j should be moved right to immediately precede the renamed S_j. Next the corresponding return of run should be move left to follow S_j.

- Subsequently, inspection moves need to be rearranged to yield a sequential play. This can be done by permuting inspection moves by O to the left through other O actions from different threads until a P-move is encountered and moving the corresponding inspection move P left to immediately follow the O move.

Then we have $h \lhd_{PO}^{*} h_2$ and, hence, $h_1(\lhd_{PO} \cup \mathcal{R})^* h_2$. $\square$

4. Soundness

To conclude, we clarify in what sense all the notions of linearisability are sound. Recall the general notion of contextual approximation (refinement) from Definition 22. In the encapsulated case libraries are being tested by clients that do not communicate with the parameter library explicitly. The corresponding definition of contextual approximation is defined below.

Definition 31 (Encapsulated $\sqsubseteq$). Given libraries $L_1, L_2 : \Psi \to \Psi'$, we write $L_1 \sqsubseteq_{\text{enc}} L_2$ when, for all $L' : \varnothing \to \Psi$ and $\Psi' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$, if $\mathsf{link}\ L' \mathbin{;} L_1\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$ then $\mathsf{link}\ L' \mathbin{;} L_2\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$.

For relational linearisability, we need yet another notion that will link $\mathcal{R}$ to contextual testing.

Definition 32. Let $\mathcal{R} \subseteq \mathcal{H}_{\Psi,\Psi'} \times \mathcal{H}_{\Psi,\Psi'}$ be a set closed under permutation of names in $\mathsf{Meths} \setminus (\Psi \cup \Psi')$. We say that a context formed by L' and $M_1, \cdots, M_N$ is $\mathcal{R}$-closed if, for any $h \in [\![\mathsf{link}\ L' \mathbin{;} -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$, $\overline{h}\, \mathcal{R}\, \overline{h'}$ implies $h' \in [\![\mathsf{link}\ L' \mathbin{;} -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$. Given $L_1, L_2 : \Psi \to \Psi'$, we write $L_1 \sqsubseteq_{\mathcal{R}} L_2$ if, for all $\mathcal{R}$-closed contexts formed from $L', M_1, \cdots, M_N$, whenever $\mathsf{link}\ L' \mathbin{;} L_1\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$ then we also have $\mathsf{link}\ L' \mathbin{;} L_2\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$.

In what follows, we shall aim to establish three correctness results:

- $L_1 \lhd L_2$ implies $L_1 \sqsubseteq L_2$,
- $L_1 \lhd_{\text{enc}} L_2$ implies $L_1 \sqsubseteq_{\text{enc}} L_2$, and
- $L_1 \lhd_{\mathcal{R}} L_2$ implies $L_1 \sqsubseteq_{\mathcal{R}} L_2$.

Finally, we note that linearisability is compatible with library composition. $\lhd$ is closed under union with libraries that use disjoint stores, while $\lhd_{\text{enc}}$ is closed under a form of sequencing that respects encapsulations (Appendix E).

4.1. Correctness

In this section we prove that the linearisability notions we introduce are correct: linearisability implies contextual approximation. The approach is based on showing that, in each case, the semantics of contexts is saturated relatively to conditions that are dual to linearisability. Hence, linearising histories does not alter the observable behaviour of a library. We start by presenting two compositionality theorems on the trace semantics, which will be used for relating library and context semantics.

4.2. Compositionality

The semantics we defined is compositional in the following ways:

- To compute the semantics of a library L inside a context C, it suffices to compose the semantics of C with that of L, for a suitable notion of context-library composition ($[\![C]\!] \oslash [\![L]\!]$).
- To compute the semantics of a union library $L_1 \cup L_2$, we can compose the semantics of L_1 and L_2, for a suitable notion of library-library composition ($[\![L_1]\!] \otimes [\![L_2]\!]$).

The above are proven using bisimulation techniques for connecting syntactic and semantic compositions, and are presented in Appendix C and Appendix D respectively.

The latter correspondence is used in Appendix E for proving that linearisability is a congruence for library composition. From the former correspondence we obtain the following result, which we shall use for showing correctness.

Theorem 33. *Let $L : \Psi \to \Psi'$, $L' : \varnothing \to \Psi, \Psi''$ and $\Psi', \Psi'' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$, with L, L' and $M_1; \cdots; M_N$ accessing pairwise disjoint parts of the store. Then:*

$$\mathsf{link}\ L' \,;\, L\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow \iff \exists h \in [\![L]\!]_N .\ \bar{h} \in [\![\mathsf{link}\ L' \,;\, -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$$

4.3. General linearisability

Recall the general notion of linearisability defined in Section 2.2, which is based on move-reorderings inside histories.

In Def.s 24 and 25 we have defined the trace semantics of libraries and contexts. The semantics turns out to be closed under $\triangleleft^*_{OP}$.

Lemma 34 (Saturation)**.** *Let $X = [\![L]\!]$ (Def. 24) or $X = [\![\mathsf{link}\ L' ; -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$ (Def. 25). Then if $h \in X$ and $h \triangleleft^*_{OP} h'$ then $h' \in X$.*

Proof. Recall that the same labelled transition system underpins the definition of X in either case. We make several observations about the single-threaded part of that system.

- The store is examined and modified only during ϵ-transitions.
- The only transition possible after a P-move is an O-move. In particular, it is never the case that a P-move is separated from the following O-move by an ϵ-transition.

Let us now consider the multi-threaded system and $t \neq t'$.

- Suppose $\rho \xRightarrow{(t',m')_P} \rho_1 \xRightarrow{\epsilon^*} \rho_2 \xRightarrow{(t,m)} \rho_3$. Then the $(t',m')_P$-transition can be delayed inside t' until after (t,m), i.e. $\rho \xRightarrow{\epsilon^*} \rho_1' \xRightarrow{(t,m)} \rho_2' \xRightarrow{(t',m')_P} \rho_3$ for some ρ_1', ρ_2'. This is possible because the ($(t',m')_P$-labelled) transition does not access or modify the store, and none of the ϵ-transitions distinguished above can be in t', thanks to our earlier observations about the behaviour of the single-threaded system.
- Analogously, suppose $\rho \xRightarrow{(t',m')} \rho_1 \xRightarrow{\epsilon^*} \rho_2 \xRightarrow{(t,m)_O} \rho_3$. Then the $(t,m)_O$-transition can be brought forward, i.e. $\rho \xRightarrow{(t,m)_O} \rho_1' \xRightarrow{(t',m')} \rho_2' \xRightarrow{\epsilon^*} \rho_3$, because it does not access or modify the store and the preceding ϵ-transitions cannot be from t. $\square$

This, along with the fact that

$$h_1 \triangleleft_{XX'} h_2 \iff h_2 \triangleleft_{X'X} h_1 \iff \overline{h_1} \triangleleft_{X'X} \overline{h_2}$$

lead us to the notion of linearisability defined in Def. 9.

We now prove the main theorem of this subsection.

```
public run;
Lock lock;
r := 0;

run = λ ().
    lock.acquire ();
    r := !r+1;
    if (!r = 1) then lock.release ();
    while (!r < 2) do ();
```

Figure 8: A library without a sequential history

Theorem 35. *$L_1 \lhd L_2$ implies $L_1 \lessapprox L_2$.*

Proof. Consider C such that $C[L_1] \Downarrow$. We need to show $C[L_2] \Downarrow$. Because $C[L_1] \Downarrow$, Theorem 33 implies that there exists $h_1 \in [\![L_1]\!]$ such that $\overline{h_1} \in [\![C]\!]$. Because $L_1 \lhd L_2$, there exists $h_2 \in [\![L_2]\!]$ with $h_1 \lhd^*_{PO} h_2$. Note that $\overline{h_1} \lhd^*_{OP} \overline{h_2}$. By Lem. 34, $\overline{h_2} \in [\![C]\!]$. Because $h_2 \in [\![L_2]\!]$ and $\overline{h_2} \in [\![C]\!]$, using Theorem 33 we can conclude $C[L_2] \Downarrow$. $\square$

Remark 36. A natural question to ask is whether the converse of Theorem 35 is true. The answer is negative and can be traced back to the fact that $\lhd$ is defined using sequential histories: in order to establish $L_1 \lhd L_2$ (for $L_1, L_2 : \Psi \to \Psi'$) one needs to identify a subset $A_2 \subseteq [\![L_2]\!] \cap \mathcal{H}^{\mathsf{seq}}_{\Psi,\Psi'}$ (i.e. consisting of *sequential* histories only) such that $L_1 \lhd A_2$.

Unfortunately, some libraries generate only non-sequential histories. We present an example of such a library, call it L, in Figure 8. Because of locks, the library from Figure 8 will only allow two threads to complete a computation. Additionally, the first thread (i.e. the one that will increment r to 1) must wait until a second thread increments the internal counter r to 2.

Observe that if L does not generate any sequential histories then we vacuously have $L \lessapprox L$, but cannot have $L \lhd L$. We conjecture that a completeness result would be possible if we allowed for non-sequential specs in the definition of $\lhd$.

4.4. Encapsulated linearisability

In this case libraries are being tested by clients that do not communicate with the parameter library explicitly. Recall from Definition 31 that, given libraries $L_1, L_2 : \Psi \to \Psi'$, we write $L_1 \lessapprox_{\text{enc}} L_2$ when, for all $L' : \varnothing \to \Psi$ and $\Psi' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$, if $\mathsf{link}\ L' \,\mathbf{;}\, L_1\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$ then $\mathsf{link}\ L' \,\mathbf{;}\, L_2\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$.

We call contexts of the above kind *encapsulated*, because the parameter library L' can no longer communicate directly with the client, unlike in Def. 22, where they shared methods in Ψ''. Consequently, $[\![\mathsf{link}\ L' \,\mathbf{;}\, -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$ can be decomposed via parallel composition into two components, whose labels correspond to $\mathcal{L}$ (parameter library) and $\mathcal{K}$ (client) respectively.

Lemma 37 (Decomposition)**.** *Suppose $L' : \varnothing \to \Psi$ and $\Psi' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$, where $\Psi \cap \Psi' = \varnothing$. Then, setting $C' \equiv \mathsf{link}\ \varnothing \,\mathbf{;}\, -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)$, we have:*

$$[\![\mathsf{link}\ L' \,\mathbf{;}\, -\ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!] = \{\, h \in \mathcal{H}^{co}_{\Psi,\Psi'} \mid (h \upharpoonright \mathcal{L}) \in [\![L']\!],\ (h \upharpoonright \mathcal{K}) \in [\![C']\!] \,\}.$$

Remark 38. Consider parameter library $L' : \varnothing \to \{m\}$ and client $\{m'\} \vdash_{\mathsf{K}} M : \mathsf{unit}$ with $m, m' \in \mathsf{Meths}_{\mathsf{unit} \to (\mathsf{unit} \to \mathsf{unit})}$, and suppose we insert in their context a "copycat" library L which implements m' as $m' = \lambda x.mx$. Then the following scenario may seem to contradict encapsulation:

- M calls $m'()$;
- L calls $m()$;
- L' returns with $m(m'')$ to L;
- and finally L copycats this return to M.

However, by definition the latter copycat is done by L returning $m'(m''')$ to M, for some *fresh* name m''', and recording internally that $m''' \mapsto \lambda x.m''x$. Hence, no methods of L' can leak to M and encapsulation holds.

Because of the above decomposition, the context semantics satisfies a stronger closure property than that already specified in Lem. 34, which in turn leads to the notion of encapsulated linearisability of Def. 13. The latter is defined in term of the symmetric reordering relation $\diamond$, which allows for swaps (in either direction) between moves from different threads if they are tagged with $\mathcal{K}$ and $\mathcal{L}$ respectively.

Moreover, we can show the following.

Lemma 39 (Encapsulated saturation)**.** *Consider* $X = [\![\mathsf{link}\ L'\,;- \ \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$ *(Definition 25). Then:*

- *If* $h \in X$ *and* $h\,(\lhd_{OP} \cup \diamond)^*\,h'$ *then* $h' \in X$.
- *Let* $s_1\,(t,x)_{OY}\,s_2\,(t,x')_{PY'}\,s_3 \in X$ *be such that no move in* s_2 *comes from thread* t*. Then* $Y = Y'$*, i.e. inside a thread only O can switch between* $\mathcal{K}$ *and* $\mathcal{L}$.

Proof. For the first claim, closure under $\lhd_{OP}$ (resp. $\diamond$) follows from Lemma 34. (resp. Lemma 37).

Suppose $h = s_1\,(t,x)_{OY}\,s_2\,(t,x')_{PY'}\,s_3$ violates the second claim and (t,x), (t,x') is the earliest such violation in h, i.e. no violations occur in s_1. Observe that then h restricted to moves of the form $(t,z)_{XY'}$ would not be alternating, which contradicts the fact that $h \upharpoonright Y'$ is a history (Lemma 37). □

Due to Theorem 33, the above property of contexts means that, in order to study termination in the encapsulated case, one can safely restrict attention to library traces satisfying a dual property to the one above, i.e. to elements of $[\![L]\!]_{\mathrm{enc}}$. Note that $[\![L]\!]_{\mathrm{enc}}$ can be obtained directly from our labelled transition system by restricting its single-threaded part to reflect the switching condition. Observe that Theorem 33 will still hold for $[\![L]\!]_{\mathrm{enc}}$ (instead of $[\![L]\!]$), because we have preserved all the histories that are compatible with context histories. We are ready to prove correctness of encapsulated linearisability.

Theorem 40. $L_1 \lhd_{\mathrm{enc}} L_2$ *implies* $L_1 \lesssim_{\mathrm{enc}} L_2$.

Proof. Similarly to Theorem 35, except we invoke Lemma 39 instead of Lemma 34. □

4.5. Relational linearisability

Finally, we examine relational linearisability (Definition 17).

Theorem 41. $L_1 \lhd_{\mathcal{R}} L_2$ *implies* $L_1 \lesssim_{\mathcal{R}} L_2$.

Proof. Consider $\mathcal{R}$-closed C such that $C[L_1]\Downarrow$. We need to show $C[L_2]\Downarrow$. Because $C[L_1]\Downarrow$, Theorem 33 implies that there exists $h_1 \in [\![L_1]\!]$ such that $\overline{h_1} \in [\![C]\!]$. Because $L_1 \lhd_{\mathcal{R}} L_2$, there exists $h_2 \in [\![L_2]\!]$ such that $h_1\,(\lhd_{PO} \cup \mathcal{R})^*\,h_2$. Because C is $\mathcal{R}$-closed by definition and closed under $\lhd_{OP}$ by Lemma 34, we have $\overline{h_2} \in [\![C]\!]$. Because $h_2 \in [\![L_2]\!]$ and $\overline{h_2} \in [\![C]\!]$, we can conclude $C[L_2]\Downarrow$. □

5. Related and future work

Linearisability has been consistently used as a correctness criterion for concurrent algorithms on a variety of data structures [15], and has inspired a variety of proof methods [16]. An explicit connection between linearisability and refinement was made in [17], where it was shown that, in base-type settings, linearisability and refinement coincide. Similar results have been proved in [18, 19, 20, 3]. Our contributions are notions of linearisability that serve as correctness criteria for libraries with methods of arbitrary order and have a similar relationship to refinement. The next natural target is to investigate proof methods for establishing linearisability of higher-order concurrent libraries. The examples proved herein are only an initial step in that direction.

At the conceptual level, [17] proposed that the verification goal behind linearisability is observational refinement. In this vein, [21] utilised logical relations as a direct method for proving refinement in a higher-order concurrent setting, while [22] introduced a program logic that builds on logical relations. On the other hand, proving conformance to a history specification has been addressed in [23] by supplying history-aware interpretations to off-the-shelf Hoare logics for concurrency. Other logic-based approaches for concurrent higher-order libraries, which do not use linearisability, include Higher-Order and Impredicative Concurrent Abstract Predicates [24, 25].

Acknowledgements. We thank the authors of [3] for bringing the higher-order linearisability problem to our attention. We would also like to thank Kasper Svendsen and Radha Jagadeesan for constructive comments. This work was partially funded by the Engineering and Physical Sciences Research Council (EP/P004172/1).

References

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

[2] R. Jagadeesan, G. Petri, C. Pitcher, J. Riely, Quarantining weakness - compositional reasoning under relaxed memory models (extended abstract), in: ESOP 2013, 2013, pp. 492–511.

[3] A. Cerone, A. Gotsman, H. Yang, Parameterised linearisability, in: Proceedings of ICALP'14, Vol. 8573 of Lecture Notes in Computer Science, Springer, 2014, pp. 98–109.

[4] J. Laird, A game semantics of Idealized CSP, in: Proceedings of MFPS'01, Elsevier, 2001, pp. 1–26, ENTCS, Vol. 45.

[5] D. R. Ghica, A. S. Murawski, Angelic semantics of fine-grained concurrency, in: Proceedings of FOSSACS, Vol. 2987 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 211–225.

[6] A. Jeffrey, J. Rathke, A fully abstract may testing semantics for concurrent objects, Theor. Comput. Sci. 338 (1-3) (2005) 17–63.

[7] J. Laird, A fully abstract trace semantics for general references, in: Proceedings of ICALP, Vol. 4596 of Lecture Notes in Computer Science, Springer, 2007, pp. 667–679.

[8] D. R. Ghica, N. Tzevelekos, A system-level game semantics, Electr. Notes Theor. Comput. Sci. 286 (2012) 191–211.

[9] D. Hendler, I. Incze, N. Shavit, M. Tzafrir, Flat combining and the synchronization-parallelism tradeoff, in: Proceedings of SPAA 2010, 2010, pp. 355–364.

[10] A. S. Murawski, N. Tzevelekos, Higher-order linearizability, in: Proceedings of CONCUR, Vol. 85 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 34:1–34:18.

[11] http://c-cube.github.io/ocaml-containers/0.21/CCMultiSet.S.html.

[12] S. Heller, M. Herlihy, V. Luchangco, M. Moir, W. N. Scherer III, N. Shavit, A lazy concurrent list-based set algorithm, in: OPODIS, 2005, pp. 3–16.

[13] P. W. O'Hearn, N. Rinetzky, M. T. Vechev, E. Yahav, G. Yorsh, Verifying linearizability with hindsight, in: PODC, 2010, pp. 85–94.

[14] S. Abramsky, G. McCusker, Game semantics, in: H. Schwichtenberg, U. Berger (Eds.), Logic and Computation, Springer-Verlag, 1998, proceedings of the 1997 Marktoberdorf Summer School.

[15] M. Moir, N. Shavit, Concurrent data structures, in: Handbook of Data Structures and Applications, Chapman and Hall/CRC, 2004.

[16] B. Dongol, J. Derrick, Verifying linearisability: A comparative survey, ACM Comput. Surv. 48 (2) (2015) 19.

[17] I. Filipovic, P. W. O'Hearn, N. Rinetzky, H. Yang, Abstraction for concurrent objects, Theor. Comput. Sci. 411 (51-52) (2010) 4379–4398.

[18] J. Derrick, G. Schellhorn, H. Wehrheim, Mechanically verified proof obligations for linearizability, ACM Trans. Program. Lang. Syst. 33 (1) (2011) 4.

[19] A. Gotsman, H. Yang, Liveness-preserving atomicity abstraction, in: Automata, Languages and Programming - 38th International Colloquium, ICALP 2011. Proceedings, Part II, 2011, pp. 453–465.

[20] H. Liang, X. Feng, Modular verification of linearizability with non-fixed linearization points, in: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13. Proceedings, 2013, pp. 459–470.

[21] A. J. Turon, J. Thamsborg, A. Ahmed, L. Birkedal, D. Dreyer, Logical relations for fine-grained concurrency, in: The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, 2013, pp. 343–356.

[22] A. Turon, D. Dreyer, L. Birkedal, Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency, in: ACM SIGPLAN International Conference on Functional Programming, ICFP'13, 2013, pp. 377–390.

[23] I. Sergey, A. Nanevski, A. Banerjee, Specifying and verifying concurrent algorithms with histories and subjectivity, in: Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015. Proceedings, 2015, pp. 333–358.

[24] K. Svendsen, L. Birkedal, Impredicative concurrent abstract predicates, in: Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014. Proceedings, 2014, pp. 149–168.

[25] K. Svendsen, L. Birkedal, M. J. Parkinson, Joins: A case study in modular specification of a concurrent reentrant higher-order library, in: ECOOP 2013 - Object-Oriented Programming - 27th European Conference. Proceedings, 2013, pp. 327–351.

Appendix

Appendix A. Big-step vs small-step reorderings

[3] defines linearisation in the general case using a "big-step" relation that applies a single permutation to the whole sequence. This contrasts with our definition as $\triangleleft_{PO}^{*}$, in which we combine multiple adjacent swaps. We show that the two definitions are equivalent.

Definition 42 ([3])**.** Let $h_1, h_2 \in \mathcal{H}_{\Psi,\Psi'}$ of equal length. We write $h_1 \triangleleft_{PO}^{\text{big}} h_2$ if there is a permutation $\pi : \{1, \cdots, |h_1|\} \to \{1, \cdots, |h_2|\}$ such that, writing $h_i(j)$ for the j-th element of h_i: for all j, we have $h_1(j) = h_2(\pi(j))$ and, for all $i < j$:

$$\begin{array}{l}((\exists t.\, h_1(i) = (t, -) \wedge h_1(j) = (t, -)) \\ \vee(\exists t_1, t_2.\, h_1(i) = (t_1, -)_P \wedge h_1(j) = (t_2, -)_O)) \implies h_2(i) < h_2(j)\end{array}$$

In other words, h_2 is obtained from h_1 by permuting moves in such a way that their order in threads is preserved and whenever a O-move occurred after an P-move in h_1, the same must apply to their permuted copies in h_2.

Lemma 43. $\triangleleft_{PO}^{\text{big}} = \triangleleft_{PO}^{*}$.

Proof. It is obvious that $\triangleleft_{PO}^{*} \subseteq \triangleleft_{PO}^{\text{big}}$, so it suffices to show the converse.

Suppose $h_1 \triangleleft_{PO}^{\text{big}} h_2$. Consider the set $X_{h_1,h_2} = \{h \mid h_1 \triangleleft_{PO}^{*} h,\, h \triangleleft_{PO}^{\text{big}} h_2\}$. Note that X_{h_1,h_2} is not empty, because $h_1 \in X_{h_1,h_2}$.

For two histories h', h'', define $\delta(h', h'')$ to be the length of the longest common prefix of h' and h''. Let $N = \max_h\{\delta(h, h_2) \mid h \in X_{h_1,h_2}\}$. Note that $N \le |h_1| = |h_2|$.

- If $N = |h_2|$ then we are done, because $N = |h_2|$ implies $h_2 \in X_{h_1,h_2}$ and, thus, $h_1 \triangleleft_{PO}^{*} h_2$.
- Suppose $N < |h_2|$ and consider h such that $N = \delta(h, h_2)$. We are going to arrive at a contradiction by exhibiting $h' \in X_{h_1,h_2}$ such that $\delta(h', h_2) > N$.

  Because $N = \delta(h, h_2)$ and $N < |h_2|$, we have

  $$\begin{array}{rcl} h_2 & = & a_1 \cdots a_N (t, m) u \\ h & = & a_1 \cdots a_N (t_1, m_1) \cdots (t_k, m_k)(t, m) u', \end{array}$$

  where $t_i \neq t$, because order in threads must be preserved. Consider

  $$h' = a_1 \cdots a_N (t, m)(t_1, m_1) \cdots (t_k, m_k) u'.$$

  Clearly $\delta(h', h_2) > N$ so, for a contradiction, it suffices to show that $h' \in X_{h_1,h_2}$. Note that because $h \triangleleft_{PO}^{\text{big}} h_2$, we must also have $h' \triangleleft_{PO}^{\text{big}} h_2$, because the new PO dependencies in h' (wrt h) caused by moving (t, m) forward are consistent with h_2. Hence, we only need to show that $h \triangleleft_{PO}^{*} h'$. Let us distinguish two cases.

  - If (t, m) is a P-move then, clearly, $h \triangleleft_{PO}^{*} h'$ (P-move moves forward).
  - If (t, m) is an O-move then, because $h \triangleleft_{PO}^{\text{big}} h_2$, all of the (t_i, m_i) actions must be O-moves (otherwise their position wrt (t, m) would have to be preserved in h_2 and it isn't). Hence, $h \triangleleft_{PO}^{*} h'$, as required.

$\square$

Appendix B. Auxiliary lemmas about histories

Recall the notions of history and history complementation (Def. 3). We next define a dual notion of history that is used for assigning semantics to contexts.

Definition 44. The set of ***co-histories*** over $\Psi \to \Psi'$ is: $\mathcal{H}^{co}_{\Psi,\Psi'} = \{\overline{h} \mid h \in \mathcal{H}_{\Psi,\Psi'}\}$.

We shall range over $\mathcal{H}^{co}_{\Psi,\Psi'}$ again using variables h, s. We can show the following.

Lemma 45. • *For all $h \in \mathcal{H}_{\Psi,\Psi'}$ we have $h \upharpoonright \mathcal{L} \in \mathcal{H}^{co}_{\varnothing,\Psi}$ and $h \upharpoonright \mathcal{K} \in \mathcal{H}_{\varnothing,\Psi'}$.*

- *For all $h \in \mathcal{H}^{co}_{\Psi,\Psi'}$ we have $h \upharpoonright \mathcal{L} \in \mathcal{H}_{\varnothing,\Psi}$ and $h \upharpoonright \mathcal{K} \in \mathcal{H}^{co}_{\varnothing,\Psi'}$.*

Lemma 46. *For any $L : \Psi \to \Psi'$, $L' : \varnothing \to \Psi, \Psi''$ and $\Psi', \Psi'' \vdash_{\mathsf{K}} M_1 \| \cdots \| M_N : \mathsf{unit}$ we have $[\![L]\!]_N \subseteq \mathcal{H}_{\Psi,\Psi'}$ and $[\![\mathsf{link}\ L'\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!] \subseteq \mathcal{H}^{co}_{\Psi,\Psi'}$.*

Proof. The relevant sequences of moves are clearly alternating and well-bracketed, when projected on single threads, because the LTS is bipartite (O- and P-configurations) and separate evaluation stacks control the evolution in each thread. Other conditions for histories follow from the partitioning of names into $\mathcal{A}_\mathcal{K}, \mathcal{A}_\mathcal{L}, \mathcal{P}_\mathcal{K}, \mathcal{P}_\mathcal{L}$ and suitable initialisation: Ψ, Ψ' are inserted into $\mathcal{A}_\mathcal{L}, \mathcal{P}_\mathcal{K}$ respectively (for $[\![L]\!]$) and into $\mathcal{P}_\mathcal{L}, \mathcal{A}_\mathcal{K}$ for $[\![C]\!]$. □

Appendix C. Trace compositionality

In this section we demonstrate how the semantics of a library inside a context can be drawn by composing the semantics of the library and that of the context. The result played a crucial role in our arguments about linearisability and contextual refinement in Section 4.1.

Let us divide (reachable) evaluation stacks into two classes: L-stacks, which can be produced in the trace semantics of a library; and C-stacks, which appear in traces of a context.

$$\begin{array}{ll} \mathcal{E}_L ::= [\,] \mid m :: E :: \mathcal{E}'_L & \qquad \mathcal{E}_C ::= [\,] \mid m :: \mathcal{E}'_C \\ \mathcal{E}'_L ::= m :: \mathcal{E}_L & \qquad \mathcal{E}'_C ::= m :: E :: \mathcal{E}_C \end{array}$$

From the trace semantics definition we have that N-configurations in the semantics of a library feature evaluation stacks of the forms $\mathcal{E}_L$ (in O-configurations) and $\mathcal{E}'_L$ (in P-configurations): these we will call ***L-stacks***. On the other hand, those produced from a context utilise ***C-stacks*** which are of the forms $\mathcal{E}_C$ (in P-configurations) and $\mathcal{E}'_C$ (in O-configurations).

From here on, when we write $\mathcal{E}$ we will mean an L-stack or a C-stack. Moreover, we will call an N-configuration ρ an ***L-configuration*** (or a ***C-configuration***), if $\rho = (\vec{\mathcal{C}}, \cdots)$ and, for each i, $\mathcal{C}_i = (\mathcal{E}_i, \cdots)$ with $\mathcal{E}_i$ an L-stack (resp. a C-stack).

Let ρ, ρ' be N-configurations and suppose $\rho = (\vec{\mathcal{C}}, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ is a C-configuration and $\rho' = (\vec{\mathcal{C}}', \mathcal{R}', \mathcal{P}', \mathcal{A}', S')$ an L-configuration. We say that ρ and ρ' are ***compatible***, written $\rho \asymp \rho'$, if S and S' have disjoint domains and, for each i:

- $\mathcal{C}_i = (\mathcal{E}_C, M)$ and $\mathcal{C}'_i = (\mathcal{E}_L, -)$, or $\mathcal{C}_i = (\mathcal{E}'_C, -)$ and $\mathcal{C}'_i = (\mathcal{E}'_L, M)$.
- If the public and abstract names of $\mathcal{C}_i$ are $(\mathcal{P}_\mathcal{L}, \mathcal{P}_\mathcal{K})$ and $(\mathcal{A}_\mathcal{L}, \mathcal{A}_\mathcal{K})$ respectively, and those of $\mathcal{C}'_i$ are $(\mathcal{P}'_\mathcal{L}, \mathcal{P}'_\mathcal{K})$ and $(\mathcal{A}'_\mathcal{L}, \mathcal{A}'_\mathcal{K})$, then $\mathcal{P}_\mathcal{L} = \mathcal{A}'_\mathcal{L}$, $\mathcal{P}_\mathcal{K} = \mathcal{A}'_\mathcal{K}$, $\mathcal{A}_\mathcal{L} = \mathcal{P}'_\mathcal{L}$ and $\mathcal{A}_\mathcal{K} = \mathcal{P}'_\mathcal{K}$.
- The private names of ρ (i.e. those in $\mathsf{dom}(\mathcal{R}) \setminus \mathcal{P}_\mathcal{L} \setminus \mathcal{P}_\mathcal{K}$) do not appear in ρ', and dually for the private names of ρ'.

- If $\mathcal{C}_i = (\mathcal{E}, \cdots)$ and $\mathcal{C}'_i = (\mathcal{E}', \cdots)$ then $\mathcal{E}$ and $\mathcal{E}'$ are in turn compatible, that is:
  - either $\mathcal{E} = m :: E :: \mathcal{E}_1$, $\mathcal{E}' = m :: \mathcal{E}'_1$ and $\mathcal{E}_1, \mathcal{E}'_1$ are compatible,
  - or $\mathcal{E} = m :: \mathcal{E}_1$, $\mathcal{E}' = m :: E :: \mathcal{E}'_1$ and $\mathcal{E}_1, \mathcal{E}'_1$ are compatible,

  or $\mathcal{E} = \mathcal{E}' = []$.

Note, in particular, that if $\rho \asymp \rho'$ then ρ must be a context configuration, and ρ' a library configuration.

We next define a trace semantics on compositions of compatible such N-configurations. We use the symbol $\oslash$ for configuration composition: we call this ***external composition***, to distinguish it from the composition of ρ and ρ' we can obtain by merging their components, which we will examine later.

$$\frac{\rho_1 \Longrightarrow \rho'_1}{\rho_1 \oslash \rho_2 \longrightarrow \rho'_1 \oslash \rho_2}\ \textsc{Int}_1 \qquad \frac{\rho_2 \Longrightarrow \rho'_2}{\rho_1 \oslash \rho_2 \longrightarrow \rho_1 \oslash \rho'_2}\ \textsc{Int}_2$$

$$\frac{\rho_1 \xRightarrow{(t, \mathsf{call}\ m(v))} \rho'_1 \quad \rho_2 \xRightarrow{(t, \mathsf{call}\ m(v))} \rho'_2}{\rho_1 \oslash \rho_2 \longrightarrow \rho'_1 \oslash \rho'_2}\ \textsc{Call}$$

$$\frac{\rho_1 \xRightarrow{(t, \mathsf{ret}\ m(v))} \rho'_1 \quad \rho_2 \xRightarrow{(t, \mathsf{ret}\ m(v))} \rho'_2}{\rho_1 \oslash \rho_2 \longrightarrow \rho'_1 \oslash \rho'_2}\ \textsc{Retn}$$

The Int rules above have side-conditions imposing that the resulting pairs of configurations are still compatible. Concretely, this means that the names created fresh in internal transitions do not match the names already present in the configurations of the other component. Note that external composition is not symmetric, due to the context/library distinction we mentioned.

Our next target is to show a correspondence between the above-defined semantic composition and the semantics obtained by (syntactically) merging compatible configurations. This will demonstrate that composing the semantics of two components is equivalent to first syntactically composing them and then evaluating the result. In order to obtain this correspondence, we need to make the semantics of syntactically composed configurations more verbose: in external composition methods belong either to the context or the library, and when e.g. the client wants to evaluate mm', with m a library method, the call is made explicit and, more importantly, m' is replaced by a fresh method name. On the other hand, when we compose syntactically such a call will be done internally, and without refreshing m'.

To counter-balance the above mismatch, we extend the syntax of terms and evaluation contexts, and the operational semantics of closed terms as follows. The semantics will now involve quadruples of the form:

$$(E[M], \mathcal{R}_1, \mathcal{R}_2, S) \text{ written also } (E[M], \vec{\mathcal{R}}, S)$$

where the two repositories correspond to context and library methods respectively, so in particular $\mathsf{dom}(\mathcal{R}_1) \cap \mathsf{dom}(\mathcal{R}_2) = \varnothing$. Moreover, inside $E[M]$ we tag method names and lambda-abstractions with indices 1 and 2 to record which of the two components (context or library) is enclosing them: the tag 1 is used for the context, and 2 for the library. Thus e.g. a name m^1 signals an occurrence of method m inside the context. Tagged methods are passed around and stored as ordinary methods, but their behaviour changes when they are applied. Moreover, we extend (tagged) evaluation contexts by explicitly marking return points of methods:

$$E ::= \bullet \mid \cdots \mid \mathsf{let}\ x = E\ \mathsf{in}\ M \mid mE \mid r := E \mid \langle m^i \rangle\, E$$

In particular, $E[M]$ may not necessarily be a (tagged) term, due to the return annotations. The new reduction rules are as follows (we omit indices when they are not used in the rules).

$$(E[i_1 \oplus i_2],\vec{\mathcal{R}},S) \to'_t (E[i],\vec{\mathcal{R}},S') \quad (i = i_1 \oplus i_2)$$
$$(E[\mathsf{tid}],\vec{\mathcal{R}},S) \to'_t (E[t],\vec{\mathcal{R}},S')$$
$$(E[\pi_j\langle v_1,v_2\rangle],\vec{\mathcal{R}},S) \to'_t (E[v_j],\vec{\mathcal{R}},S')$$
$$(E[\mathsf{if}\ i\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1],\vec{\mathcal{R}},S) \to'_t (E[M_j],\vec{\mathcal{R}},S) \ \ (j = (i > 0))$$
$$(E[\lambda^i x.M],\vec{\mathcal{R}},S) \to'_t (E[m^i],\vec{\mathcal{R}} \uplus_i (m \mapsto \lambda x.M),S)$$
$$(E[m^i v],\vec{\mathcal{R}},S) \to'_t (E[M\{v/x\}^i],\vec{\mathcal{R}},S) \quad \text{if } \mathcal{R}_i(m) = \lambda x.M$$
$$(E[m^i v],\vec{\mathcal{R}},S) \to'_t (E[\langle m^i\rangle\, M\{v'/x\}^{3-i}],\vec{\mathcal{R}}',S) \quad \text{if } \mathcal{R}_{3-i}(m) = \lambda x.M \text{ with}$$
$$\qquad \mathsf{Meths}(v) = \{m_1,\cdots,m_k\}, v' = v\{\,m'_j/m_j \mid 1 \le j \le k\}, \vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_i \{m'_j \mapsto \lambda y.m_j y \mid 1 \le j \le k\}$$
$$(E[\langle m^i\rangle\, v],\vec{\mathcal{R}},S) \to'_t (E[v'^i],\vec{\mathcal{R}} \uplus_{3-i} \{m'_j \mapsto \lambda y.m_j y\},S) \text{ with } m_j, m'_j \text{ and } v' \text{ as above}$$
$$(E[\mathsf{let}\ x = v\ \mathsf{in}\ M],\vec{\mathcal{R}},S) \to'_t (E[M\{v/x\}],\vec{\mathcal{R}},S)$$
$$(E[!r],\vec{\mathcal{R}},S) \to'_t (E[S(r)],\vec{\mathcal{R}},S)$$
$$(E[r := i],\vec{\mathcal{R}},S) \to'_t (E,\vec{\mathcal{R}},S[r \mapsto i])$$
$$(E[r := m^i],\vec{\mathcal{R}},S) \to'_t (E,\vec{\mathcal{R}},S[r \mapsto m^i])$$

Above we write M^i for the term M with all its methods and lambdas tagged (or re-tagged) with i. Moreover, we use the convention e.g. $\vec{\mathcal{R}} \uplus_1 (m \mapsto \lambda x.M) = (\mathcal{R}_1 \uplus (m \mapsto \lambda x.M),\mathcal{R}_2)$. Note that the repositories need not contain tags as, whenever a method is looked up, we subsequently tag its body explicitly.

Thus, the computationally observable difference of the new semantics is in the rule for reducing $E[m^i v]$ when m is not in the domain of $\mathcal{R}_i$: this corresponds precisely to the case where e.g. a library method is called by the context with another method as argument. A similar behaviour is exposed when such a method is returning. However, this novelty merely adds fresh method names by η-expansions and does not affect the termination of the reduction.

Defining parallel reduction $\Longrightarrow'$ in an analogous way to $\Longrightarrow$, we can show the following. We let a quadruple $(M_1\|\cdots\|M_N,\mathcal{R},S)$ be *final* if $M_i = ()$ for all i, and we write $(M_1\|\cdots\|M_N,\mathcal{R},S)\Downarrow$ if $(M_1\|\cdots\|M_N,\mathcal{R},S)$ can reduce to some final quadruple; these notions are defined for $(M_1\|\cdots\|M_N,\mathcal{R}_1,\mathcal{R}_2,S)$ in the same manner.

Lemma 47. *For any legal $(M_1\|\cdots\|M_N,\mathcal{R}_1,\mathcal{R}_2,S)$, we have that $(M_1\|\cdots\|M_N,\mathcal{R}_1,\mathcal{R}_2,S)\Downarrow$ iff $(M_1\|\cdots\|M_N,\mathcal{R}_1 \cup \mathcal{R}_2,S)\Downarrow$.*

We now proceed to syntactic composition of N-configurations. Given a pair $\rho_1 \asymp \rho_2$, we define a single quadruple corresponding to their syntactic composition, called their ***internal composition***, as follows. Let $\rho_1 = (\vec{\mathcal{C}},\mathcal{R}_1,\mathcal{P}_1,\mathcal{A}_1,S_1)$ and $\rho_2 = (\vec{\mathcal{C}}',\mathcal{R}_2,\mathcal{P}_2,\mathcal{A}_2,S_2)$ and, for each i, $\mathcal{C}_i = (\mathcal{E}_i,X_i)$ and $\mathcal{C}'_i = (\mathcal{E}'_i,X'_i)$, with $\{X_i,X'_i\} = \{M_i,-\}$, and we let $k_i = 1$ just if $X_i = M_i$. We let the internal composition of ρ_1 and ρ_2 be the quadruple:

$$\rho_1 \mathbin{\wedge\!\!\!\wedge} \rho_2 = ((\mathcal{E}_1 \mathbin{\wedge\!\!\!\wedge} \mathcal{E}'_1)[M_1^{k_1}]\|\cdots\|(\mathcal{E}_N \mathbin{\wedge\!\!\!\wedge} \mathcal{E}'_N)[M_N^{k_N}],\mathcal{R}_1,\mathcal{R}_2,S_1 \uplus S_2)$$

where compatible evaluation stacks $\mathcal{E},\mathcal{E}'$ are composed into a single evaluation context

$\mathcal{E} \curlywedge \mathcal{E}'$, as follows.

$$
\begin{aligned}
(m :: E :: \mathcal{E}) \curlywedge (m :: \mathcal{E}') &= (\mathcal{E} \curlywedge \mathcal{E}')[E[\langle m \rangle \bullet]^1] \\
(m :: \mathcal{E}') \curlywedge (m :: E :: \mathcal{E}) &= (\mathcal{E} \curlywedge \mathcal{E}')[E[\langle m \rangle \bullet]^2]
\end{aligned}
$$

and $[\,] \curlywedge [\,] = \bullet$. Unfolding the above, we have that, for example:

$$
\begin{aligned}
&[m_k, E_k, m_{k-1}, m_{k-2}, E_{k-2}, \cdots, m_1, E_1] \\
&\curlywedge [m_k, m_{k-1}, E_{k-1}, m_{k-2}, \cdots, m_1] = E_1^1[\langle m_1^1 \rangle\, E_2^2[\cdots E_k^{k'}[\langle m_k^{k'} \rangle \bullet]\cdots]]
\end{aligned}
$$

where $k' = 2 - (k \mod 2)$.

We proceed to fleshing out the correspondence. We observe that an L-configuration ρ can be the final configuration of a trace just if all its components are O-configurations with empty evaluation stacks. On the other hand, for C-configurations, we need to reach P-configurations with terms (). Thus, we call an N-configuration ρ *final* if $\rho = (\vec{\mathcal{C}}, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ and either $\mathcal{C}_i = ([\,], -)$ for all i, or $\mathcal{C}_i = ([\,], ())$ for all i.

Let us write $(\mathcal{S}_1, \hookrightarrow_1, \mathcal{F}_1)$ for the transition system induced from external composition, and $(\mathcal{S}_2, \hookrightarrow_2, \mathcal{F}_2)$ be the transition system derived from internal composition:

- $\mathcal{S}_1 = \{\rho \oslash \rho' \mid \rho \asymp \rho'\}$, $\mathcal{F}_1 = \{\rho \oslash \rho' \in \mathcal{S}_1 \mid \rho, \rho' \text{ final}\}$, and $\hookrightarrow_1$ the transition relation $\longrightarrow$ defined previously.
- $\mathcal{S}_2 = \{(M_1 \| \cdots \| M_N, \vec{\mathcal{R}}, S) \mid (M_1 \| \cdots \| M_N, \mathcal{R}_1 \uplus \mathcal{R}_2, S) \text{ valid}\}$, $\mathcal{F}_2 = \{x \in \mathcal{S}_2 \mid x \text{ final}\}$, and $\hookrightarrow_2$ the transition relation $\Longrightarrow'$ defined above.

A relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is called a *bisimulation* if, for all $(x_1, x_2) \in R$:

- $x_1 \in \mathcal{F}_1$ iff $x_2 \in \mathcal{F}_2$,
- if $x_1 \hookrightarrow_1 x_1'$ then $x_2 \hookrightarrow_2 x_2'$ and $(x_1', x_2') \in R$,
- if $x_2 \hookrightarrow_2 x_2'$ then $x_1 \hookrightarrow_1 x_1'$ and $(x_1', x_2') \in R$.

Given $(x_1, x_2) \in \mathcal{S}_1 \times \mathcal{S}_2$, we say that x_1 and x_2 are *bisimilar*, written $x_1 \sim x_2$, if $(x_1, x_2) \in R$ for some bisimulation R.

Lemma 48. *Let $\rho \asymp \rho'$ be compatible N-configurations. Then, $(\rho \oslash \rho') \sim (\rho \curlywedge \rho')$.*

Recall we write $\bar{h}$ for the O/P complement of the history h. We can now prove Theorem 33, which states that the behaviour of a library L inside a context C can be deduced by composing the semantics of L and C.

Theorem 33 *Let $L : \Psi \to \Psi'$, $L' : 1 \to \Psi, \Psi_1$ and $\Psi', \Psi_1 \vdash M_1, \cdots, M_N : \mathsf{unit}$, with L, L' and $M_1; \cdots; M_N$ accessing pairwise disjoint parts of the store. Then,* $\mathsf{link}\ L' \,\mathbf{;}\, L\ \mathsf{in}\ (M_1 \| \cdots \| M_N) \Downarrow$ *iff there is $h \in [\![L]\!]_N$ such that $\bar{h} \in [\![\mathsf{link}\ L' \,\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)]\!]$.*

Proof. Let C be the context $\mathsf{link}\ L' \,\mathbf{;} - \mathsf{in}\ (M_1 \| \cdots \| M_N)$, and suppose $(L) \longrightarrow_{\mathrm{lib}}^* (\epsilon, \mathcal{R}_0, S_0)$ and $(L') \longrightarrow_{\mathrm{lib}}^* (\epsilon, \mathcal{R}_0', S_0')$ with $\mathsf{dom}(\mathcal{R}_0) \cap \mathsf{dom}(\mathcal{R}_0') = \mathsf{dom}(S_0) \cap \mathsf{dom}(S_0') = \varnothing$. We set:

$$
\begin{aligned}
\rho_0 &= (([\,], -) \| \cdots \| ([\,], -), \mathcal{R}_0, (\varnothing, \Psi'), (\Psi, \varnothing), S_0) \\
\rho_0' &= (([\,], M_1) \| \cdots \| ([\,], M_N), \mathcal{R}_0', (\Psi, \varnothing), (\varnothing, \Psi'), S_0')
\end{aligned}
$$

We pick these as the initial N-configurations for $[\![L]\!]_N$ and $[\![C]\!]$ respectively. Moreover, we have that $(L' \,\mathbf{;}\, L) \longrightarrow_{\mathrm{lib}}^* (\epsilon, \mathcal{R}_0'', S_0'')$ where $\mathcal{R}_0'' = \{(m, (\mathcal{R}_0 \uplus \mathcal{R}_0')(m)\{!\vec{r}/\vec{m}\}) \mid$

$m \in \mathsf{dom}(\mathcal{R}_0 \uplus \mathcal{R}'_0)\}$ and $S''_0 = (S_0 \uplus S'_0)\{!\vec{r}/\vec{m}\} \uplus_{\mathsf{s}} \{(r_i, m_i) \mid i = 1, \cdots, n\}$, assuming $\Psi = \{m_1, \cdots, m_n\}$ and $r_1, \cdots, r_n$ are fresh references of corresponding types. Hence, the initial triple for $[\![C[L]]\!]$ is taken to be $\phi_0 = (([], M_1)\|\cdots\|([], M_N), \mathcal{R}''_0, S''_0)$. On the other hand, $\rho'_0 \curlywedge \rho_0 = (([], M_1)\|\cdots\|([], M_N), \mathcal{R}'_0, \mathcal{R}_0, S_0 \uplus S'_0)$ and, using also Lemma 47, we have that $\phi_0 \Downarrow$ iff $\rho'_0 \curlywedge \rho_0 \Downarrow$.

Then, for the forward direction of the claim, from $\phi_0 \Downarrow$ we obtain that $\rho'_0 \curlywedge \rho_0 \Downarrow$. From the previous lemma, we have that so does $\rho'_0 \oslash \rho_0$. From the latter reduction we obtain the required common history. Conversely, suppose $h \in [\![L]\!]_N$ and $\bar{h} \in [\![C]\!]$. WLOG, assume that $\mathsf{Meths}(h) \cap (\mathsf{dom}(\mathcal{R}_0) \cup \mathsf{dom}(\mathcal{R}'_0)) \subseteq \Psi \cup \Psi_1 \cup \Psi'$ (we can appropriately alpha-covert $\mathcal{R}_0$ and $\mathcal{R}'_0$ for this). Then, ρ_0 and ρ'_0 both produce h, with opposite polarities. By definition of the external composite reduction, we then have that $\rho'_0 \oslash \rho_0$ reduces to some final state. By the previous lemma, we have that $\rho'_0 \curlywedge \rho_0$ reduces to some final quadruple, which in turn implies that $\phi_0 \Downarrow$, i.e. $\mathsf{link}\ L' \,;\, L\ \mathsf{in}\ (M_1\|\cdots\|M_N) \Downarrow$. $\square$

We conclude this section with the proofs of the last two lemmata used.

Appendix C.1. Proof of Lemma 47

We purpose to show that, for any legal $(M_1\|\cdots\|M_N, \mathcal{R}_1, \mathcal{R}_2, S)$, $(M_1\|\cdots\|M_N, \mathcal{R}_1, \mathcal{R}_2, S) \Downarrow$ iff $(M_1\|\cdots\|M_N, \mathcal{R}_1 \cup \mathcal{R}_2, S) \Downarrow$.

We prove something stronger. For any repository $\mathcal{R}$ whose entries are of the form $(m, \lambda x.m'x)$, we define a directed graph $\mathcal{G}(\mathcal{R})$ where vertices are all methods appearing in $\mathcal{R}$, and (m, m') is a (directed) edge just if $\mathcal{R}(m) = \lambda x.m'x$. In such a case, we call $\mathcal{R}$ an ***expansion class*** if $\mathcal{G}(\mathcal{R})$ is acyclic and all its vertices have at most one outgoing edge.Moreover, given an expansion class $\mathcal{R}$, we define the method-for-method substitution $\{\mathcal{R}\}$ that assigns to each vertex m of $\mathcal{G}(\mathcal{R})$ the (unique) leaf m' such that there is a directed path from m to m' in $\mathcal{G}(\mathcal{R})$. Let us write $\mathcal{L}(\mathcal{R})$ for the set of leaves of $\mathcal{G}(\mathcal{R})$. For any quadruple $\phi = (E_1[M_1]\|\cdots\|E_N[M_N], \mathcal{R}_1, \mathcal{R}_2, S)$ and expansion class $\mathcal{R} \subseteq \mathcal{R}_1 \cup \mathcal{R}_2$, we define the triple:

$$\begin{aligned}
\phi^{\#\mathcal{R}} &= (\underline{E_1[M_1]}\|\cdots\|\underline{E_N[M_N]}, \mathcal{R}_1 \cup \mathcal{R}_2, S)\{\mathcal{R}\} \\
&= (\underline{E_1[M_1]}\{\mathcal{R}\}\|\cdots\|\underline{E_N[M_N]}\{\mathcal{R}\}, (\mathcal{R}_1 \cup \mathcal{R}_2)\{\mathcal{R}\}, S\{\mathcal{R}\})
\end{aligned}$$

where $\mathcal{R}'\{\mathcal{R}\} = \{(m, \mathcal{R}'(m)\{\mathcal{R}\}) \mid m \in \mathsf{dom}(\mathcal{R}' \setminus \mathcal{R}) \cup \mathcal{L}(\mathcal{R})\}$, $S\{\mathcal{R}\} = (S \upharpoonright \mathsf{Refs}_{\mathsf{int}}) \cup \{(r, S(r)\{\mathcal{R}\}) \mid r \in \mathsf{dom}(S) \setminus \mathsf{Refs}_{\mathsf{int}}\}$, and $\underline{E[M]}$ is the term obtained from $E[M]$ by removing all tagging.

We next define a notion of indexed bisimulation between the transition systems produced from quadruples and triples respectively. Given an expansion class $\mathcal{R}$, a relation $R_{\mathcal{R}}$ between quadruples and triples is called an *$\mathcal{R}$-bisimulation* if, whenever $\phi_1 R_{\mathcal{R}} \phi_2$:

- ϕ_1 final implies ϕ_2 final
- ϕ_2 final implies $\phi_2 \Downarrow$
- $\phi_1 \Longrightarrow' \phi'_1$ implies $\phi_2 \Longrightarrow^{=} \phi'_2$ and $\phi'_1 R_{\mathcal{R}'} \phi'_2$ for some expansion class $\mathcal{R}' \supseteq \mathcal{R}$
- $\phi_2 \Longrightarrow \phi'_2$ implies $\phi_1 \Longrightarrow'^{*} \phi'_1$ and $\phi'_1 R_{\mathcal{R}'} \phi'_2$ for some expansion class $\mathcal{R}' \supseteq \mathcal{R}$.

Thus, Lemma 47 directly follows from the next result.

Lemma 49. *For all expansion classes $\mathcal{R}$, the relation $R_{\mathcal{R}}$ =*

$$\{(\phi, \phi^{\#\mathcal{R}}) \mid \phi = (E_1[M_1] \| \cdots \| E_N[M_N], \vec{\mathcal{R}}, S) \textit{ legal } \wedge \mathcal{R} \subseteq \mathcal{R}_1 \cup \mathcal{R}_2\}$$

is a bisimulation.

Proof. Suppose $\phi R_{\mathcal{R}} \phi^{\#\mathcal{R}}$. We note that finality conditions are satisfied: if ϕ is final then so is $\phi^{\#\mathcal{R}}$; while if $\phi^{\#\mathcal{R}}$ is final then all its contexts are from the grammar:

$$E' ::= \bullet \mid \langle m^i \rangle\, E'$$

so $\phi \Downarrow$ by acyclicity of $\mathcal{G}(\mathcal{R})$.
Suppose now $\phi \Longrightarrow \phi'$, say due to $(E_1[M_1], \mathcal{R}_1, \mathcal{R}_2, S) \to_1' (E_1'[M_1'], \mathcal{R}_1', \mathcal{R}_2', S')$. In case the reduction is not a function call or return, then it can be clearly simulated by $\phi^{\#\mathcal{R}}$. Otherwise, suppose:

- $(E_1[m^i v], \vec{\mathcal{R}}, S) \to_1' (E_1[M\{v/x\}^i], \vec{\mathcal{R}}, S)$. If $m \notin \mathsf{dom}(\mathcal{R})$ then, writing $\mathcal{R}_{12}$ for $\mathcal{R}_1 \cup \mathcal{R}_2$, the above can be simulated by $(\underline{E_1}[mv], \mathcal{R}_{12}, S)\{\mathcal{R}\} \to_1 (\underline{E_1}[M\{v/x\}], \mathcal{R}_{12}, S)\{\mathcal{R}\}$. If, on the other hand, $m \in \mathsf{dom}(\mathcal{R})$, suppose $\mathcal{R}_i(m) = \lambda x.m'x$, then $M = m'x$ and $m\{\mathcal{R}\} = m'\{\mathcal{R}\}$ so we have:

  $$\underline{E_1[M\{v/x\}^i]}\{\mathcal{R}\} = \underline{E_1[(m'v)^i]}\{\mathcal{R}\} = \underline{E_1[(mv)^i]}\{\mathcal{R}\}$$

  and $E_1[(mv)^i] = E_1[m^i v]$ by the way the semantics was defined, so $\phi'^{\#\mathcal{R}} = \phi^{\#\mathcal{R}}$.

- $(E_1[m^i v], \vec{\mathcal{R}}, S) \to_1' (E_1[\langle m^i \rangle\, M\{v'/x\}^{3-i}], \vec{\mathcal{R}}', S)$, with $\mathcal{R}_{3-i}(m) = \lambda x.M$, $\mathsf{Meths}(v) = \{m_1, \cdots, m_k\}$, $v' = \{\vec{m}'/\vec{m}\}$ and $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_i \{m_j' \mapsto \lambda x.m_j x \mid 1 \leq j \leq k\}$. Let $\mathcal{R}' = \mathcal{R} \uplus \{m_j' \mapsto \lambda x.m_j x \mid 1 \leq j \leq k\} \subseteq \mathcal{R}_1' \cup \mathcal{R}_2'$. If $m \notin \mathsf{dom}(\mathcal{R})$ then $(\underline{E_1}[mv], \mathcal{R}_{12}, S)\{\mathcal{R}\} \to_1 (\underline{E_1}[M\{v/x\}], \mathcal{R}_{12}, S)\{\mathcal{R}\}$, and we have:

  $$\begin{aligned} \underline{E_1[\langle m^i \rangle\, M\{v'/x\}^{3-i}]}\{\mathcal{R}'\} &= \underline{E_1}[M\{v'/x\}]\{\mathcal{R}'\} \\ &= \underline{E_1}[M\{v/x\}]\{\mathcal{R}\} \end{aligned}$$

  Moreover, $\mathcal{R}_{12}\{\mathcal{R}\} = (\mathcal{R}_1' \cup \mathcal{R}_2')\{\mathcal{R}'\}$ and $S\{\mathcal{R}\} = S\{\mathcal{R}'\}$, so $\phi'^{\#\mathcal{R}} = (\underline{E_1}[M\{v/x\}], \mathcal{R}_{12}, S)\{\mathcal{R}\}$. On the other hand, if $\mathcal{R}(m) = \lambda x.m''x$ then:

  $$\begin{aligned} \underline{E[\langle m^i \rangle\, M\{v'/x\}^{3-i}]}\{\mathcal{R}'\} &= \underline{E}[m''v']\{\mathcal{R}'\} \\ &= \underline{E}[m''v]\{\mathcal{R}\} = \underline{E}[mv]\{\mathcal{R}\} \end{aligned}$$

  so $\phi^{\#\mathcal{R}} = \phi'^{\#\mathcal{R}'}$.

- Finally, the cases for method-return reductions are treated similarly as above.

Suppose now $\phi^{\#\mathcal{R}} \Longrightarrow \phi'$, where recall that we write ϕ as $(E_1[M_1] \| \cdots \| E_N[M_N], \vec{\mathcal{R}}, S)$. We show by induction on $\mathsf{size}_{\mathcal{R}}(E_1[M_1], \cdots, E_N[M_N])$ that $\phi \Longrightarrow' \phi''$ and $\phi' R_{\mathcal{R}'} \phi''$ for some $\mathcal{R}' \supseteq \mathcal{R}$. The size-function we use measures the length of $\mathcal{G}(\mathcal{R})$-paths that appear inside its arguments:

$$\begin{aligned} \mathsf{size}_{\mathcal{R}}(E_1[M_1], \cdots, E_N[M_N]) &= \mathsf{size}_{\mathcal{R}}(E_1[M_1]) + \cdots + \mathsf{size}_{\mathcal{R}}(E_N[M_N]) \\ \mathsf{size}_{\mathcal{R}}(E[M]) &= \sum_{m \in X_1} 2|m|_{\mathcal{R}} + \sum_{m \in X_2} 1 \end{aligned}$$

where X_1 is the multiset containing all occurrences of methods $m \in \mathsf{dom}(\mathcal{R})$ inside $E[M]$ in call position (e.g. mM'), and X_2 contains all occurrences of methods $m \in \mathsf{dom}(\mathcal{R})$ inside $E[M]$ in return position (i.e. $\langle m^i \rangle \cdots$). We write $|m|_{\mathcal{R}}$ for the length of the unique directed path from m to a leaf in $\mathcal{G}(\mathcal{R})$. The fact that X_1, X_2 are multisets reflects that we count all occurrences of m in call/return positions. Suppose WLOG that the reduction to ϕ' is due to some $(\underline{E_1[M_1]}, \mathcal{R}_{12}, S)\{\mathcal{R}\} \to_1 (E'[M'], \mathcal{R}', S')$. If the reduction happens inside $\underline{M_1}\{\mathcal{R}\}$ (this case also encompasses the base case of the induction) then the only case we need to examine is that of the reduction being a method call. In such a case, suppose we have $\underline{E_1[M_1]\{\mathcal{R}\}} = E[mv]$, $E' = E$, $M' = M\{v/x\}$ and $\mathcal{R}_{12}\{\mathcal{R}\}(m) = \lambda x.M$. Then, $E_1[M_1] = \tilde{E}[\tilde{m}^i \tilde{v}]$ for some $\tilde{E}, \tilde{m}, \tilde{v}$ such that $\tilde{m}\{\mathcal{R}\} = m$, $\tilde{v}\{\mathcal{R}\} = v$ and $\underline{\tilde{E}}\{\mathcal{R}\} = E$. If $m \not\equiv \tilde{m}$ then, supposing $\mathcal{R}(\tilde{m}) = \lambda x.\tilde{m}' x$ we have the following cases:

- $(\tilde{E}[\tilde{m}^i \tilde{v}], \vec{\mathcal{R}}, S) \to_1' (\tilde{E}[\tilde{m}'^i \tilde{v}], \vec{\mathcal{R}}, S) = \phi_1''$
- $(\tilde{E}[\tilde{m}^i \tilde{v}], \vec{\mathcal{R}}, S) \to_1' (\tilde{E}[\langle \tilde{m}^i \rangle (\tilde{m}' v')^{3-i}], \vec{\mathcal{R}}', S) = \phi_1''$, with $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_{3-i} \{m_j' \mapsto \lambda x.m_j x \mid 1 \le j \le k\}$, etc.

Let ϕ'' be the extension of ϕ_1'' to an N-quadruple by using the remaining $E_i[M_i]$'s of ϕ, so that $\phi \Longrightarrow' \phi''$. In the first case above we have that $\phi''^{\#\mathcal{R}} = \phi$, and in the latter that $\phi''^{\#\mathcal{R}'} = \phi$ (with $\mathcal{R}' = \mathcal{R} \uplus \{m_j' \mapsto \lambda x.m_j x \mid 1 \le j \le k\}$), and we appeal to the IH. Suppose now that $\tilde{m} = m$ and $\mathcal{R}_{12}(m) = \lambda x.\tilde{M}$. Then, one of the following is the case:

- $(\tilde{E}[\tilde{m}^i \tilde{v}], \vec{\mathcal{R}}, S), \vec{\mathcal{R}}, S) \to_1' (\tilde{E}[\tilde{M}\{\tilde{v}/x\}^i], \vec{\mathcal{R}}, S) = \phi_1''$
- $(\tilde{E}[\tilde{m}^i \tilde{v}], \vec{\mathcal{R}}, S) \to_1' (\tilde{E}[\langle \tilde{m}^i \rangle \tilde{M}\{v'/x\}^{3-i}], \vec{\mathcal{R}}', S) = \phi_1''$, with $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_{3-i} \{m_j' \mapsto \lambda x.m_j x \mid 1 \le j \le k\}$, etc.

Extending ϕ_1'' to ϕ'' as above, in the former case we then have that $\phi''^{\#\mathcal{R}} = \phi'$, and in the latter that $\phi''^{\#\mathcal{R}'} = \phi'$, as required.
Finally, let us suppose that M_1 is some value v. Then, we can write E_1 as $E_1 = E_2[E']$, with E' coming from the grammar $E' ::= \bullet \mid \langle m^i \rangle E'$ and E_2 not being of the form $E''[\langle m^i \rangle \bullet]$. Observe that $\underline{E_1} = \underline{E_2}$. If $E' = \bullet$ then by a case analysis on E_1 we can see that $\phi^{\#\mathcal{R}}$ can simulate the reduction. Otherwise, $(E_2[E'[v]], \vec{\mathcal{R}}, S) \to_1' (E_2[E''[v'^i]], \vec{\mathcal{R}}', S)$ whereby $E' = E''[\langle m^i \rangle \bullet]$ and $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_{3-i} \{m_j' \mapsto \lambda x.m_j x \mid 1 \le j \le k\}$, etc. We have that

$$\phi_1'' = (\underline{E_2[E''[v'^i]]}, \vec{\mathcal{R}}', S)\{\mathcal{R}'\} = (\underline{E_2[E'[v]]}, \vec{\mathcal{R}}, S)\{\mathcal{R}\}$$

and hence, extending ϕ_1'' to ϕ'', we have $\phi''^{\#\mathcal{R}'} = \phi^{\#\mathcal{R}}$. We can now appeal to the IH. □

Appendix C.2. . Proof of Lemma 48

Let $\rho \asymp \rho'$ be compatible N-configurations. Then, $(\rho \oslash \rho') \sim (\rho \curlywedge \rho')$.

We prove that the relation $R = \{(\rho_1 \oslash \rho_2, \rho_1 \curlywedge \rho_2) \mid \rho_1 \asymp \rho_2\}$ is a bisimulation. Let us suppose that $(\rho_1 \oslash \rho_2, \rho_1 \curlywedge \rho_2) \in R$.

- Suppose $\rho_1 \oslash \rho_2 \hookrightarrow_1 \rho_1' \oslash \rho_2'$. If the transition is due to (INT1) then $\rho_2 = \rho_2'$ and we can see that $\rho_1 \curlywedge \rho_2 \Longrightarrow' \rho_1' \curlywedge \rho_2$. Similarly if the transition is due to (INT2). Suppose now we used instead (CALL), e.g. $\rho_1 \xRightarrow{(1,\mathsf{call}\, m(v))} \rho_1'$ and $\rho_2 \xRightarrow{(1,\mathsf{call}\, m(v))} \rho_2'$, and let us consider the case where $v \in \mathsf{Meths}$ (the other case is simpler). Then, assuming

$\rho_1 = (\mathcal{C}_1^1 \| \cdots, \mathcal{R}_1, \mathcal{P}_1, \mathcal{A}_1, S_1)$ and $\rho_2 = (\mathcal{C}_1^2 \| \cdots, \mathcal{R}_2, \mathcal{P}_2, \mathcal{A}_2, S_2)$, we have that either of the following scenarios holds, for some $\mathtt{x} \in \{\mathcal{K}, \mathcal{L}\}$: $\mathcal{C}_1^1 = (\mathcal{E}_1, E[mm'])$, $\mathcal{C}_1^2 = (\mathcal{E}_2, -)$ and

$$\begin{aligned}
&(\mathcal{E}_1, E[mm'], \mathcal{R}_1, \mathcal{P}_1, \mathcal{A}_1, S_1) \xrightarrow{\mathsf{call}\, m(v)}_1 \\
&\qquad (m :: E :: \mathcal{E}_1, \mathcal{R}_1 \uplus (v \mapsto \lambda x.m'x), \mathcal{P}_1 \cup_{\mathtt{x}} \{v\}, \mathcal{A}_1, S_1) \\
&(\mathcal{E}_2, -, \mathcal{R}_2, \mathcal{P}_2, \mathcal{A}_2, S_2) \xrightarrow{\mathsf{call}\, m(v)}_1 \\
&\qquad\qquad (m :: \mathcal{E}_2, M\{v/x\}, \mathcal{R}_2, \mathcal{P}_1, \mathcal{A}_1 \cup_{\mathtt{x}} \{v\}, S_2)
\end{aligned}$$

or its dual, where ρ_2 contains the code initiating the call. Focusing WLOG in the former case and setting $S = S_1 \uplus S_2$:

$$\begin{aligned}
\rho_1 \barwedge \rho_2 &= ((\mathcal{E}_1 \barwedge \mathcal{E}_2)[E[m^1 m']] \| \cdots, \mathcal{R}_1, \mathcal{R}_2, S) \\
&\hookrightarrow_2 ((\mathcal{E}_1 \barwedge \mathcal{E}_2)[E[\langle m^1 \rangle\, M\{v/x\}^2]] \| \cdots, \mathcal{R}_1', \mathcal{R}_2, S) \\
&= \rho_1' \barwedge \rho_2' \quad (\mathcal{R}_1' = \mathcal{R}_1 \uplus (v \mapsto \lambda x.m'x))
\end{aligned}$$

The case for (RETN) is treated similarly.

- Suppose $\rho_1 \barwedge \rho_2 = (E[M_1] \| M_2 \| \cdots \| M_N, \vec{\mathcal{R}}, S) \hookrightarrow_2 (E[M_1'] \| M_2 \| \cdots \| M_N, \vec{\mathcal{R}}', S')$ and let $\rho_1 = ((\mathcal{E}_1, M_1'') \| \cdots, \mathcal{R}_1, \mathcal{P}_1, \mathcal{A}_1, S_1)$ and $\rho_2 = ((\mathcal{E}_2, -) \| \cdots, \mathcal{R}_2, \mathcal{P}_2, \mathcal{A}_2, S_2)$, where $(\mathcal{E}_1 \barwedge \mathcal{E}_2)[M_1''] = E[M_1]$. If the redex M_1 is not of the forms $M_1 = m^1 v$ or $M_1 = \langle m^1 \rangle\, v$, with $m \in \mathsf{dom}(\mathcal{R}_2)$, then the reduction can clearly be simulated by $\rho_1 \oslash \rho_2$ (internally, by ρ_1). Otherwise, similarly as above, the reduction can be simulated by a mutual call/return of m.

Finally, it is clear that $\rho_1 \oslash \rho_2$ is final iff $\rho_1 \barwedge \rho_2$ is final. □

Appendix D. Library Compositionality

This compositionality result will allow us to compose histories of component libraries in order to obtain those of their composite library. Let $L_1 : \Psi_1 \to \Psi_2$ and $L_2 : \Psi_1' \to \Psi_2'$. The semantic composition will be guided by two sets of names Π, P. Π contains method names that are shared between by the respective libraries and their context. Thus $\Pi \supseteq \Psi_1 \cup \Psi_1' \cup \Psi_2 \cup \Psi_2'$. The names in P, on the other hand, will be used for private communication between L_1 and L_2. Consequently, $\Pi \cap \mathrm{P}$ consists of names that can be used both for internal communication between L_1 and L_2, and for contextual interactions, i.e. $\Pi \cap \mathrm{P} = (\Psi_1 \cup \Psi_1') \cap (\Psi_2 \cup \Psi_2')$.

Given $h_i \in [\![L_i]\!] (i = 1, 2)$, we define the *composition* of h_1 and h_2, written $h_1 \barwedge^{\sigma}_{\Pi,\mathrm{P}} h_2$, as a partial operation depending on Π, P and an additional parameter $\sigma \in \{0, 1, 2\}^*$ which we call a *scheduler*. It is given inductively as follows. We let $\epsilon \barwedge^{\epsilon}_{\Pi,\mathrm{P}} \epsilon = \epsilon$ and:

$$\begin{aligned}
(t, \mathsf{call}\, m(v)) s_1 \barwedge^{0\sigma}_{\Pi,\mathrm{P}} (t, \mathsf{call}\, m(v)) s_2 &= s_1 \barwedge^{\sigma}_{\Pi,\mathrm{P}'} s_2 \\
(t, \mathsf{ret}\, m(v)) s_1 \barwedge^{0\sigma}_{\Pi,\mathrm{P}} (t, \mathsf{ret}\, m(v)) s_2 &= s_1 \barwedge^{\sigma}_{\Pi,\mathrm{P}'} s_2 \\
(t, \mathsf{call}\, m(v))_{PY} s_1 \barwedge^{1\sigma}_{\Pi,\mathrm{P}} s_2 &= (t, \mathsf{call}\, m(v))_{PY} (s_1 \barwedge^{\sigma}_{\Pi',\mathrm{P}} s_2) \\
(t, \mathsf{ret}\, m(v))_{PY} s_1 \barwedge^{1\sigma}_{\Pi,\mathrm{P}} s_2 &= (t, \mathsf{ret}\, m(v))_{PY} (s_1 \barwedge^{\sigma}_{\Pi',\mathrm{P}} s_2) \\
(t, \mathsf{call}\, m(v))_{OY} s_1 \barwedge^{1\sigma}_{\Pi,\mathrm{P}} s_2 &= (t, \mathsf{call}\, m(v))_{OY} (s_1 \barwedge^{\sigma}_{\Pi',\mathrm{P}} s_2) \\
(t, \mathsf{ret}\, m(v))_{OY} s_1 \barwedge^{1\sigma}_{\Pi,\mathrm{P}} s_2 &= (t, \mathsf{ret}\, m(v))_{OY} (s_1 \barwedge^{\sigma}_{\Pi',\mathrm{P}} s_2)
\end{aligned}$$

along with the dual rules for the last four cases (i.e. where we schedule 2 in each case). Note that the definition uses sequences of moves that are suffixes of histories (such as s_i). The above equations are subject to the following side conditions:

- $\mathsf{Meths}(v) \cap (\Pi \cup \mathrm{P}) = \varnothing$, $\Pi' = \Pi \uplus \mathsf{Meths}(v)$ and $\mathrm{P}' = \mathrm{P} \uplus \mathsf{Meths}(v)$;
- $m \in \mathrm{P}$ in the 0-scheduling cases;
- $m \in \Pi$ in the 1-scheduling cases and, also, $m \in \Pi \smallsetminus \mathrm{P}$ in the third case (the P-call);
- in the 1-scheduling cases, we also require that the leftmost move with thread index t in s_2 is not a P-move.

History composition is a partial function: if the conditions above are not met, or h_1, h_2, σ are not of the appropriate form, then the composition is undefined. The above conditions ensure that the composed histories are indeed compatible and can be produced by composing actual libraries. For instance, the last condition corresponds to determinacy of threads: there can only be at most one component starting with a P-move in each thread t. We then have the following correspondence.

Theorem 50. *If $L_1 : \Psi_1 \to \Psi_2$ and $L_2 : \Psi_1' \to \Psi_2'$ access disjoint parts of the store then*

$$[\![L_1 \cup L_2]\!]_N = \{\, h \in \mathcal{H} \mid \exists \sigma, h_1 \in [\![L_1]\!]_N, h_2 \in [\![L_2]\!]_N.\ h = h_1 \curlywedge^{\sigma}_{\Pi_0,\mathrm{P}_0} h_2 \}$$

with $\Pi_0 = \Psi_1 \cup \Psi_2 \cup \Psi_1' \cup \Psi_2'$ and $\mathrm{P}_0 = (\Psi_1 \cup \Psi_1') \cap (\Psi_2 \cup \Psi_2')$.

The rest of this section is devoted in proving the Theorem.

Recall that we examine library composition in the sense of union of libraries. This scenario is more general than the one of Appendix C as, during composition via union, the calls and returns of each of the component libraries may be caught by the other library or passed as a call/return to the outer context. Thus, the setting of this section comprises given libraries $L_1 : \Psi_1 \to \Psi_2$ and $L_2 : \Psi_1' \to \Psi_2'$, such that $\Psi_2 \cap \Psi_2' = \varnothing$, and relating their semantics to that of their union $L_1 \cup L_2 : (\Psi_1 \cup \Psi_1') \smallsetminus (\Psi_2 \cup \Psi_2') \to \Psi_2 \cup \Psi_2'$.

Given configurations for L_1 and L_2, in order to be able to reduce them together we need to determine which of their methods can be used for communication between them, and which for interacting with the external context, which represents player O in the game. We will therefore employ a set of method names, denoted by Π and variants, to register those methods used for interaction with the external context. Another piece of information we need to know is in which component in the composition was the last call played, or whether it was an internal call instead. This is important so that, when O (or P) has the choice to return to both components, in the same thread, we know which one was last to call and therefore has precedence. We use for this purpose sequences $w = (w_1, \cdots, w_N)$ where, for each i, $w_i \in \{0,1,2\}^*$. Thus, if e.g. $w_1 = 2w_1'$, this would mean that, in thread 1, the last call to O, was done from the second component; if, on the other hand, $w_1 = 0w_1'$ then the last call in thread 1 was an internal one between the two components. Given such a w and some $j \in \{0,1,2\}$, for each index t, we write $j +_t w$ for $w[t \mapsto (jw_t)]$.

Let us fix libraries $L_1 : \Psi_1 \to \Psi_2$ and $L_2 : \Psi_1' \to \Psi_2'$. Let ρ_1, ρ_2 be N-configurations, and in particular L-configurations, and suppose that $\rho_1 = (\vec{\mathcal{C}}, \mathcal{R}, \mathcal{P}, \mathcal{A}, S)$ and $\rho_2 = (\vec{\mathcal{C}}', \mathcal{R}', \mathcal{P}', \mathcal{A}', S')$. Moreover, let $\Psi_1 \cup \Psi_2 \cup \Psi_1' \cup \Psi_2' \subseteq \Pi$. We say that ρ_1 and ρ_2 are (w, Π)***-compatible***, written $\rho_1 \asymp^w_\Pi \rho_2$, if S, S' have disjoint domains and, for each i;

- $\mathcal{C}_i = (\mathcal{E}_L', M)$ and $\mathcal{C}_i' = (\mathcal{E}_L, -)$, or $\mathcal{C}_i = (\mathcal{E}_L, -)$ and $\mathcal{C}_i' = (\mathcal{E}_L', M)$, or $\mathcal{C}_i = (\mathcal{E}_{L1}, -)$ and $\mathcal{C}_i' = (\mathcal{E}_{L2}, -)$.

- We have $\Psi_1 \subseteq \mathcal{A}_l$, $\Psi_2 \subseteq \mathcal{P}_\mathcal{K}$, $\Psi_1' \subseteq \mathcal{A}_\mathcal{L}'$, $\Psi_2' \subseteq \mathcal{P}_\mathcal{K}'$ and, setting

$$\mathrm{P} = (\mathcal{P}_\mathcal{K} \cap \mathcal{A}_\mathcal{L}') \uplus (\mathcal{P}_\mathcal{L} \cap \mathcal{A}_\mathcal{K}') \uplus (\mathcal{P}_\mathcal{K}' \cap \mathcal{A}_\mathcal{L}) \uplus (\mathcal{P}_l' \cap \mathcal{A}_\mathcal{K})$$

  we also have:

  - $(\mathcal{P}_\mathcal{L} \uplus \mathcal{P}_\mathcal{K} \uplus \mathcal{A}_l \uplus \mathcal{A}_\mathcal{K}) \cap (\mathcal{P}_\mathcal{L}' \uplus \mathcal{P}_\mathcal{K}' \uplus \mathcal{A}_l' \uplus \mathcal{A}_\mathcal{K}') = \mathrm{P} \uplus (\Psi_1 \cap \Psi_1')$,
  - $\Pi \cap \mathrm{P} = (\Psi_2 \cup \Psi_2') \cap (\Psi_1 \cup \Psi_1')$,
  - $\Pi \cup \mathrm{P} = \mathcal{P}_\mathcal{L} \cup \mathcal{P}_\mathcal{K} \cup \mathcal{P}_l' \cup \mathcal{P}_\mathcal{K}' \cup \mathcal{A}_\mathcal{L} \cup \mathcal{A}_\mathcal{K} \cup \mathcal{A}_\mathcal{L}' \cup \mathcal{A}_\mathcal{K}'$.
- The private names of $\mathcal{R}$ do not appear in ρ_2, and dually for the private names of $\mathcal{R}'$.
- If $\mathcal{C}_i = (\mathcal{E}, \cdots)$ and $\mathcal{C}_i' = (\mathcal{E}', \cdots)$ then $\mathcal{E}$ and $\mathcal{E}'$ are *w_i-compatible*, that is, either $\mathcal{E} = \mathcal{E}' = []$ or:

  - $\mathcal{E} = m :: \mathcal{E}_1$ and $\mathcal{E}' \in \mathcal{E}_L$, with $m \in \Pi$, $w_i = 1u$ and $\mathcal{E}_1, \mathcal{E}'$ are u-compatible,
  - or $\mathcal{E} = m :: \mathcal{E}_1$ and $\mathcal{E}' = m :: E :: \mathcal{E}_2$, with $m \in \mathrm{P}$, $w_i = 0u$ and $\mathcal{E}_1, \mathcal{E}_2$ are u-compatible,
  - or $\mathcal{E} = m :: E :: \mathcal{E}_1$ and $\mathcal{E}' \in \mathcal{E}_L$, with $m \in \Pi \setminus \mathrm{P}$, $w_i = 1u$ and $\mathcal{E}_1, \mathcal{E}'$ are u-compatible,

  or the dual of one of the three conditions above holds.

Given $\rho_1 \asymp_\Pi^w \rho_2$, we let their external composition be denoted as $\rho_1 \otimes_\Pi^w \rho_2$ (and note that now the notation is symmetric for ρ_1 and ρ_2) and define the semantics for external composition by these rules:

$$\frac{\rho_1 \Longrightarrow \rho_1'}{\rho_1 \otimes_\Pi^w \rho_2 \longrightarrow \rho_1' \otimes_\Pi^w \rho_2}\ \textsc{Int}_1$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{call}\ m(v))} \rho_1' \quad \rho_2 \xRightarrow{(t,\mathsf{call}\ m(v))} \rho_2'}{\rho_1 \otimes_\Pi^w \rho_2 \longrightarrow \rho_1' \otimes_\Pi^{0+_t w} \rho_2'}\ \textsc{Call}\ (m \in \mathrm{P})$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{ret}\ m(v))} \rho_1' \quad \rho_2 \xRightarrow{(t,\mathsf{ret}\ m(v))} \rho_2'}{\rho_1 \otimes_\Pi^{0+_t w} \rho_2 \longrightarrow \rho_1' \otimes_\Pi^w \rho_2'}\ \textsc{Retn}\ (m \in \mathrm{P})$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{call}\ m(v))_{PY}} \rho_1'}{\rho_1 \otimes_\Pi^w \rho_2 \xrightarrow{(t,\mathsf{call}\ m(v))_{PY}} \rho_1' \otimes_{\Pi'}^{1+_t w} \rho_2}\ \textsc{PCall}_1\ (m \in \Pi \setminus \mathrm{P})$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{ret}\ m(v))_{PY}} \rho_1'}{\rho_1 \otimes_\Pi^{1+_t w} \rho_2 \xrightarrow{(t,\mathsf{ret}\ m(v))_{PY}} \rho_1' \otimes_{\Pi'}^w \rho_2}\ \textsc{PRetn}_1\ (m \in \Pi)$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{call}\ m(v))_{OY}} \rho_1'}{\rho_1 \otimes_\Pi^w \rho_2 \xrightarrow{(t,\mathsf{call}\ m(v))_{OY}} \rho_1' \otimes_{\Pi'}^{1+_t w} \rho_2}\ \textsc{OCall}_1\ (m \in \Pi)$$

$$\frac{\rho_1 \xRightarrow{(t,\mathsf{ret}\ m(v))_{OY}} \rho_1'}{\rho_1 \otimes_\Pi^{1+_t w} \rho_2 \xrightarrow{(t,\mathsf{ret}\ m(v))_{OY}} \rho_1' \otimes_{\Pi'}^w \rho_2}\ \textsc{ORetn}_1\ (m \in \Pi \setminus \mathrm{P})$$

along with their dual counterparts ($\textsc{Int}_2$, $\textsc{XCall}_2$, $\textsc{XRetn}_2$). The internal rules above have the same side-conditions on name privacy as before. Moreover, in ($\textsc{XRetn}i$) and

(XCALLi), for X=O,P, we let $\Pi' = \Pi \uplus_t \mathsf{Meths}(v)$ and impose that the t-th component of ρ_{3-i} be an O-configuration and $\mathsf{Meths}(v) \cap \mathsf{Meths}(\rho_{3-i}) = \varnothing$.

We can now show the following.

Lemma 51. *Let $\rho_1 \asymp^w_\Pi \rho_2$ and suppose $\rho_1 \otimes^w_\Pi \rho_2 \xrightarrow{s}{}^* \rho'_1 \otimes^{w'}_{\Pi'} \rho'_2$ for some sequence s of moves. Then, $\rho'_1 \asymp^{w'}_{\Pi'} \rho'_2$.*

We next juxtapose the semantics of external composition to that obtained by internally composing the libraries and then deriving the multi-threaded semantics of the result. As before, we call the latter form *internal composition*. The traces we obtain are produced from a transition relation, written $\Longrightarrow'$, between configurations of the form $(\mathcal{C}_1\|\cdots\|\mathcal{C}_N, \mathcal{R}_1, \mathcal{R}_2, \mathcal{P}, \mathcal{A}, S)$, also written $(\vec{\mathcal{C}}, \vec{\mathcal{R}}, \mathcal{P}, \mathcal{A}, S)$. In particular, in each $\mathcal{C}_i = (\mathcal{E}_i, X_i)$ with $X_i = E_i[M_i]$ or $X_i = -$, E_i is selected from the extended evaluation contexts and $\mathcal{E}_i$ is an *extended L-stack*, that is, of either of the following two forms:

$$\mathcal{E}_{\mathsf{ext}} ::= [\,] \mid m^i :: E :: \mathcal{E}'_{\mathsf{ext}} \qquad \mathcal{E}'_{\mathsf{ext}} ::= m :: \mathcal{E}_{\mathsf{ext}}$$

where E is again from the extended evaluation contexts.

First, given u-compatible evaluation stacks $\mathcal{E}, \mathcal{E}'$, we construct a pair $\mathcal{E} \curlywedge^u \mathcal{E}'$ consisting of an extended evaluation context and an extended L-stack, as follows. Given $\mathcal{E} \curlywedge^u \mathcal{E}' = (E', \mathcal{E}'')$:

$$\begin{aligned}
&(m :: E :: \mathcal{E}) \curlywedge^{0u} (m :: \mathcal{E}') = (E'[E[\langle m\rangle \bullet]^1], \mathcal{E}'')\\
&(m :: \mathcal{E}) \curlywedge^{0u} (m :: E :: \mathcal{E}') = (E'[E[\langle m\rangle \bullet]^2], \mathcal{E}'')\\
&(m :: \mathcal{E}) \curlywedge^{1u} \mathcal{E}' = \mathcal{E} \curlywedge^{2u} (m :: \mathcal{E}') = (\bullet, m :: E' :: \mathcal{E}'')\\
&(m :: E :: \mathcal{E}) \curlywedge^{1u} \mathcal{E}' = \mathcal{E} \curlywedge^{2u} (m :: E :: \mathcal{E}')\\
&\qquad\qquad\qquad\qquad\qquad = (\bullet, m :: E'[E] :: \mathcal{E}'') \ \text{ if } \mathcal{E}' \in \mathcal{E}_L
\end{aligned}$$

and $[\,] \curlywedge^\epsilon [\,] = (\bullet, [\,])$.

For each pair $\rho_1 \asymp^w_\Pi \rho_2$, we define a configuration corresponding to their syntactic composition as follows. Let $\rho_1 = (\mathcal{C}_1\|\cdots\|\mathcal{C}_N, \mathcal{R}_1, \mathcal{P}_1, \mathcal{A}_1, S_1)$ and $\rho_2 = (\mathcal{C}'_1\|\cdots\|\mathcal{C}'_N, \mathcal{R}_2, \mathcal{P}_2, \mathcal{A}_2, S_2)$ and, for each i, $\mathcal{C}_i = (\mathcal{E}_i, X_i)$ and $\mathcal{C}'_i = (\mathcal{E}'_i, X'_i)$. If $\mathcal{E}_i \curlywedge^u \mathcal{E}'_i = (E_i, \mathcal{E}''_i)$, we set:

$$\mathcal{C}_i \curlywedge^u \mathcal{C}'_i = \begin{cases} (\mathcal{E}''_i, E_i[M^1]) & \text{if } X_i = M \text{ and } X'_i = -\\ (\mathcal{E}''_i, E_i[M^2]) & \text{if } X_i = - \text{ and } X'_i = M\\ (\mathcal{E}''_i, -) & \text{if } X_i = X'_i = - \end{cases}$$

We then let the internal composition of ρ_1 and ρ_2 be:

$$\rho_1 \curlywedge^w_\Pi \rho_2 = (\mathcal{C}_1 \curlywedge^{w_1} \mathcal{C}'_1\|\cdots\|\mathcal{C}_N \curlywedge^{w_N} \mathcal{C}'_N, \mathcal{R}_1, \mathcal{R}_2, \mathcal{P}', \mathcal{A}', S_1 \uplus S_2)$$

where we set $\mathcal{P}' = ((\mathcal{P}_{1\mathcal{L}} \uplus \mathcal{P}_{2\mathcal{L}}) \cap \Pi, (\mathcal{P}_{1\mathcal{K}} \uplus \mathcal{P}_{2\mathcal{K}}) \cap \Pi)$ and $\mathcal{A}' = ((\mathcal{A}_{1\mathcal{L}} \cup \mathcal{A}_{2\mathcal{L}}) \cap (\Pi \setminus \mathrm{P}), (\mathcal{A}_{1\mathcal{K}} \uplus \mathcal{A}_{2\mathcal{K}}) \cap \Pi)$.

Now, as expected, the definition of $\Longrightarrow'$ builds upon $\to'_t$. The definition of the latter is given by the following rules.

$$\frac{(E[M],\vec{\mathcal{R}},S) \to'_t (E'[M'],\vec{\mathcal{R}}',S')}{(\mathcal{E},E[M],\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \to'_t (\mathcal{E},E'[M'],\vec{\mathcal{R}}',\mathcal{P},\mathcal{A},S')}\ (\textsc{Int}')$$

$$(\mathcal{E},E[m^i v],\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xrightarrow{\mathsf{call}\ m(v')_{PY}}{}'_t (m^i :: E :: \mathcal{E},-,\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A},S) \qquad (\mathrm{PC}\textsc{y}')$$

$$(m :: \mathcal{E},v,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xrightarrow{\mathsf{ret}\ m(v')_{PY}}{}'_t (\mathcal{E},-,\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A},S) \qquad (\mathrm{PR}\textsc{y}')$$

$$(\mathcal{E},-,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xrightarrow{\mathsf{call}\ m(v)_{OY}}{}'_t (m :: \mathcal{E},M\{v/x\}^i,\vec{\mathcal{R}},\mathcal{P},\mathcal{A}',S) \qquad (\mathrm{OC}\textsc{y}')$$

$$(m^i :: E :: \mathcal{E},-,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xrightarrow{\mathsf{ret}\ m(v)_{OY}}{}'_t (\mathcal{E},E[v^i],\vec{\mathcal{R}},\mathcal{P},\mathcal{A}',S) \qquad (\mathrm{OR}\textsc{y}')$$

The side-conditions are similar to those for the relation $\to_t$ between ordinary configurations, with the following exceptions: in (PC$\textsc{y}'$), if $\mathsf{Meths}(v) = \{m_1,\cdots,m_k\}$ then $v' = v\{m'_j/m_j \mid 1 \le j \le k\}$, for fresh m'_j's, and $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_i \{m'_j \mapsto \lambda x.m_j x\}$; and in (PR$\textsc{y}'$), if $m \in \mathsf{dom}(\mathcal{R}_i)$ then $\vec{\mathcal{R}}' = \vec{\mathcal{R}} \uplus_i \{m'_j \mapsto \lambda x.m_j x\}$, etc. Moreover, in (OC$\textsc{y}'$) we have that $m \in \mathsf{dom}(\mathcal{R}_i)$. Finally, we let

$$(\vec{\mathcal{C}},\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xRightarrow{(t,x)_{XY}}{}' (\vec{\mathcal{C}}[t \mapsto \mathcal{C}'],\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A}',S')$$

just if $(\mathcal{C}_t,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S) \xrightarrow{x_{XY}}{}'_t (\mathcal{C}',\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A}',S')$.

We next relate the transition systems induced by external (via $\otimes$) and internal composition (via $\curlywedge$). Let us write $(\mathcal{S}_1,\hookrightarrow_1,\mathcal{F}_1)$ for the transition system induced by external composition of compatible N-configurations (so $\hookrightarrow_1$ is $\longrightarrow$), and $(\mathcal{S}_2,\hookrightarrow_2,\mathcal{F}_2)$ be the one for internal composition (so $\hookrightarrow_2$ is $\Longrightarrow'$). Finality of extended N-configurations $(\mathcal{C}_1\|\cdots\|\mathcal{C}_N,\vec{\mathcal{R}},\cdots)$ is defined as expected: all $\mathcal{C}_i$'s must be $([],-)$. A relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is called a *bisimulation* if, for all $(x_1,x_2) \in R$:

- $x_1 \in \mathcal{F}_1$ iff $x_2 \in \mathcal{F}_2$,
- if $x_1 \hookrightarrow_1 x'_1$ then $x_2 \hookrightarrow_2 x'_2$ and $(x'_1,x'_2) \in R$,
- if $x_1 \xhookrightarrow{(t,x)_{XY}}_1 x'_1$ then $x_2 \xhookrightarrow{(t,x)_{XY}}_2 x'_2$ and $(x'_1,x'_2) \in R$,
- if $x_2 \hookrightarrow_2 x'_2$ then $x_1 \hookrightarrow_1 x'_1$ and $(x'_1,x'_2) \in R$,
- if $x_2 \xhookrightarrow{(t,x)_{XY}}_2 x'_2$ then $x_1 \xhookrightarrow{(t,x)_{XY}}_1 x'_1$ and $(x'_1,x'_2) \in R$.

Again, we say that x_1 and x_2 are *bisimilar*, and write $x_1 \sim x_2$, if there exists a bisimulation R such that $(x_1,x_2) \in R$.

Lemma 52. *Let $\rho \asymp^w_\Pi \rho'$ be compatible N-configurations. Then, $(\rho \otimes^w_\Pi \rho') \sim (\rho \curlywedge^w_\Pi \rho')$.*

Proof. We prove that the relation $R = \{(\rho_1 \otimes^w_\Pi \rho_2, \rho_1 \curlywedge^w_\Pi \rho_2) \mid \rho_1 \asymp^w_\Pi \rho_2\}$ is a bisimulation. Let us suppose that $(\rho_1 \otimes^w_\Pi \rho_2, \rho_1 \curlywedge^w_\Pi \rho_2) \in R$.

- Let $\rho_1 \otimes^w_\Pi \rho_2 \xrightarrow{(t,x)} \rho'_1 \otimes^{w'}_{\Pi'} \rho'_2$ with the transition being due to (XC$\textsc{all}_1$), e.g. $\rho_1 \xRightarrow{(1,\mathsf{call}\ m(v))} \rho'_1$ and $\rho'_2 = \rho_2$, $w' = 1 +_1 w$ and $\Pi' = \Pi \uplus_1 \mathsf{Meths}(v)$, $\mathsf{Meths}(v) = \{m'_1,\cdots,m'_j\}$, and recall that $\mathsf{Meths}(v) \cap \mathsf{Meths}(\rho_2) = \varnothing$. Then, assuming $\rho_1 =$

$(\mathcal{C}_1^1\|\cdots,\mathcal{R}_1,\mathcal{P}_1,\mathcal{A}_1,S_1)$, we have that one of the following holds, for some $\mathsf{x}\in\{\mathcal{K},\mathcal{L}\}$:

$$\mathcal{C}_1^1=(\mathcal{E}_1,E[mv'])\ \text{ and }\ (\mathcal{C}_1^1,\mathcal{R}_1,\mathcal{P}_1,\mathcal{A}_1,S_1)\xrightarrow{\mathsf{call}\ m(v)}_1$$
$$(m::E::\mathcal{E}_1,-,\mathcal{R}_1\uplus\{m_j'\mapsto\lambda x.m_jx\mid 1\le j\le k\},\mathcal{P}_1\cup_{\mathsf{x}}\mathsf{Meths}(v),\mathcal{A}_1,S_1)$$
$$\mathcal{C}_1^1=(\mathcal{E}_1,-)\ \text{ and }\ (\mathcal{C}_1^1,\mathcal{R}_1,\mathcal{P}_1,\mathcal{A}_1,S_1)\xrightarrow{\mathsf{call}\ m(v)}_1$$
$$(m^1::\mathcal{E}_1,mv,\mathcal{R}_1,\mathcal{P}_1,\mathcal{A}_1\cup_{\mathsf{x}}\mathsf{Meths}(v),S_1)$$

In the former case, if $\rho_2=((\mathcal{E}_2,-)\|\cdots,\mathcal{R}_2,\mathcal{P}_2,\mathcal{A}_2,S_2)$ with $\mathcal{E}_1\mathbin{\wedge\!\!\wedge}^{w_1}\mathcal{E}_2=(E',\mathcal{E})$, we get:

$$\rho_1\mathbin{\wedge\!\!\wedge}_\Pi^w\rho_2=((\mathcal{E},E'[E[mv']^1])\|\cdots,\mathcal{R}_1,\mathcal{R}_2,\mathcal{P},\mathcal{A},S)$$
$$\xRightarrow{(1,\mathsf{call}\ m(v))}{}'$$
$$(m^1::E'[E^1]::\mathcal{E},-)\|\cdots,\mathcal{R}_1\uplus\{m_j'\mapsto\lambda x.m_jx\mid 1\le j\le k\},\mathcal{R}_2,\mathcal{P}',\mathcal{A},S)$$

with $\mathcal{P},\mathcal{A}$ as in the definition of composition and $\mathcal{P}'=\mathcal{P}\cup_{\mathsf{x}}\mathsf{Meths}(v)$, and the latter N-configuration equals $\rho_1'\mathbin{\wedge\!\!\wedge}_{\Pi'}^{w'}\rho_2$. The other case is treated in the same manner, and we work similarly for (RETN$_1$).

- On the other hand, if the transition is due to (CALL) or (RETN) then we work as in the proof of Lemma 48.

- Suppose $\rho_1\mathbin{\wedge\!\!\wedge}_\Pi^w\rho_2=(\mathcal{C}_1\|\cdots,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S)\xRightarrow{(1,\mathsf{call}\ m(v))}{}'(\mathcal{C}_1'\|\cdots,\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A}',S)$. Then, assuming WLOG that $v\in\mathsf{Meths}$, one of the following must be the case, for some $\mathsf{x}\in\{\mathcal{K},\mathcal{L}\}$ and $i\in\{1,2\}$:

$$\mathcal{C}_1=(\mathcal{E},E[m^iv'])\ \text{ and }\ (\mathcal{C}_1,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S)\xrightarrow{\mathsf{call}\ m(v)}{}_1'$$
$$(m^i::E::\mathcal{E},\vec{\mathcal{R}}\uplus_i\{m_j'\mapsto\lambda x.m_jx\mid 1\le j\le k\},\mathcal{P}\cup_{\mathsf{x}}\mathsf{Meths}(v),\mathcal{A},S)$$
$$\mathcal{C}_1(\mathcal{E},-)\ \text{ and }\ (\mathcal{C}_1,\vec{\mathcal{R}},\mathcal{P},\mathcal{A},S)\xrightarrow{\mathsf{call}\ m(v)}{}_1'$$
$$(m::\mathcal{E},M\{v/x\}^i,\vec{\mathcal{R}},\mathcal{P},\mathcal{A}\cup_{\mathsf{x}}\mathsf{Meths}(v),S)$$

We only examine the former case, as the latter one is similar, and suppose that $i=1$. Taking $\rho_j=(\mathcal{C}_1^j\|\cdots,\mathcal{R}_j,\mathcal{P}_j,\mathcal{A}_j,S_i)$, for $j=1,2$, we have that $(\mathcal{C}_1^1,\mathcal{C}_1^2)=((\mathcal{E}_1,E'[mv'],(\mathcal{E}_2,-))$, for some $E,\mathcal{E}_1,\mathcal{E}_2$ such that $\mathcal{E}_1\mathbin{\wedge\!\!\wedge}^{w_1}\mathcal{E}_2=(E'',\mathcal{E})$ and $E=E''[E'^1]$. Moreover, taking $\mathcal{R}_1'=\mathcal{R}_1\uplus\{m_j'\mapsto\lambda x.m_jx\mid 1\le j\le k\}$, $\mathcal{P}_1'=\mathcal{P}_1\uplus_{\mathsf{x}}\{v\}$, $w'=1+_1w$ and $\Pi'=\Pi\uplus\mathsf{Meths}(v)$ (note $\mathsf{Meths}(v)=\{m_1',\cdots,m_k'\}$),

$$\rho_1\otimes_\Pi^w\rho_2\xrightarrow{(1,\mathsf{call}\ m(v))}((m::E'::\mathcal{E}_1,-)\|\cdots,\mathcal{R}_1',\mathcal{P}_1',\mathcal{A}_1,S_1)\otimes_{\Pi'}^{w'}\rho_2=\rho_1'\otimes_{\Pi'}^{w'}\rho_2$$

and $\rho_1'\mathbin{\wedge\!\!\wedge}_{\Pi'}^{w'}\rho_2=(\mathcal{C}_1'\|\cdots,\vec{\mathcal{R}}',\mathcal{P}',\mathcal{A}',S)$ as required. The case for return transitions is similar.

- On the other hand, if the transition out of $\rho_1\mathbin{\wedge\!\!\wedge}_\Pi^w\rho_2$ does not have a label then we work as in the proof of Lemma 48.

Moreover, by definition of syntactic composition, $\rho_1\otimes_\Pi^w\rho_2$ is final iff $\rho_1\mathbin{\wedge\!\!\wedge}_\Pi^w\rho_2$ is. $\square$

Given an N-configuration ρ and a history h, let us write $\rho \Downarrow h$ if $\rho \xRightarrow{h} \rho'$ for some final configuration ρ'. Similarly if ρ is of the form $(\vec{C}, \vec{\mathcal{R}}, \mathcal{P}, \mathcal{A}, S)$. We have the following connections in history productions. The next lemma is proven in a similar fashion as Lemma 47.

Lemma 53. *For any legal $(M_1\|\cdots\|M_N, \mathcal{R}_1, \mathcal{R}_2, \mathcal{P}, \mathcal{A}, S)$ and history h, we have that $(M_1\|\cdots\|M_N, \mathcal{R}_1, \mathcal{R}_2, \mathcal{P}, \mathcal{A}, S) \Downarrow h$ iff $(M_1\|\cdots\|M_N, \mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{P}, \mathcal{A}, S) \Downarrow h$.*

Lemma 54. *For any compatible N-configurations $\rho_1 \asymp^w_\Pi \rho_2$ and history h, $(\rho_1 \otimes^w_\Pi \rho_2) \Downarrow h$ iff:*

$$\exists h_1, h_2, \sigma.\ \rho_1 \Downarrow h_1 \wedge \rho_2 \Downarrow h_2 \wedge h = h_1 \curlywedge^{\sigma}_{\Pi,\mathrm{P}} h_2$$

where P is computed from ρ_1, ρ_2 and Π as before.

Proof. We show that, for any compatible N-configurations $\rho_1 \asymp^w_\Pi \rho_2$ and history suffix s, $(\rho_1 \otimes^w_\Pi \rho_2) \Downarrow s$ iff:

$$\exists s_1, s_2, \sigma.\ \rho_1 \Downarrow s_1 \wedge \rho_2 \Downarrow s_2 \wedge s = s_1 \curlywedge^{\sigma}_{\Pi,\mathrm{P}} s_2$$

where P is computed from ρ_1, ρ_2 and Π as in the beginning of this section.

The left-to-right direction follows from straightforward induction on the length of the reduction that produces s. For the right-to-left direction, we do induction on the length of σ. If $\sigma = \epsilon$ then $s_1 = s_2 = s = \epsilon$. Otherwise, we do a case analysis on the first element of σ. We only look at the most interesting subcase, namely of $\sigma = 0\sigma'$. Then, for some $m \in \mathrm{P}$:

$$s_1 = (t, \mathsf{call}\, m(v))s_1' \qquad s_2 = (t, \mathsf{call}\, m(v))s_2'$$

By $\rho_i \Downarrow s_i$ and $\rho_1 \asymp^w_\Pi \rho_2$ we have that $\rho_1 \otimes^w_\Pi \rho_2 \longrightarrow \rho_1' \otimes^{w'}_\Pi \rho_2$, where $w' = 0 +_t w$ and $\rho_1' \asymp^{w'}_\Pi \rho_2'$. Also, $\rho_i' \Downarrow s_i'$ and $s = s_1' \curlywedge^{\sigma'}_{\Pi,\mathrm{P}'} s_2'$ so, by IH, $(\rho_1' \otimes^{w'}_\Pi \rho_2') \Downarrow s$. ☐

We can now prove the correspondence between the traces of component libraries and those of their union.

Theorem 50 *Let $L_1 : \Psi_1 \to \Psi_2$ and $L_2 : \Psi_1' \to \Psi_2'$ be libraries accessing disjoint parts of the store. Then,*

$$[\![L_1 \cup L_2]\!]_N = \{h \in \mathcal{H}^L \mid \exists \sigma, h_1 \in [\![L_1]\!]_N, h_2 \in [\![L_2]\!]_N.\ h = h_1 \curlywedge^{\sigma}_{\Pi_0,\mathrm{P}_0} h_2\}$$

with $\Pi_0 = \Psi_1 \cup \Psi_2 \cup \Psi_1' \cup \Psi_2'$ and $\mathrm{P}_0 = (\Psi_1 \cup \Psi_1') \cap (\Psi_2 \cup \Psi_2')$.

Proof. Let us suppose $(L_i) \longrightarrow^*_{\mathrm{lib}} (\epsilon, \mathcal{R}_i, S_i)$, for $i = 1, 2$, with $\mathsf{dom}(\mathcal{R}_1) \cap \mathsf{dom}(\mathcal{R}_2) = \mathsf{dom}(S_1) \cap \mathsf{dom}(S_2) = \varnothing$. We set:

$$\begin{aligned}\rho_1 &= (([], -)\|\cdots\|([], -), \mathcal{R}_1, (\varnothing, \Psi_2), (\Psi_1, \varnothing), S_1)\\ \rho_2 &= (([], -)\|\cdots\|([], -), \mathcal{R}_2, (\varnothing, \Psi_2'), (\Psi_1', \varnothing), S_2)\end{aligned}$$

We pick these as the initial configurations for $[\![L_1]\!]_N$ and $[\![L_2]\!]_N$ respectively. Then, $(L_1 \cup L_2) \longrightarrow^*_{\mathrm{lib}} (\epsilon, \mathcal{R}_0, S_0)$ where $\mathcal{R}_0 = \mathcal{R}_1 \uplus \mathcal{R}_2$ and $S_0 = S_1 \uplus S_2$, and we take

$$\rho_0 = (([], -)\|\cdots\|([], -), \mathcal{R}_0, (\varnothing, \Psi_2 \cup \Psi_2'), ((\Psi_1 \cup \Psi_1') \setminus \mathrm{P}_0, \varnothing), S_0)$$

as the initial N-configuration for $[\![L_1 \cup L_2]\!]_N$. On the other hand, we have $\rho_1 \curlywedge^{\epsilon}_{\Pi_0} \rho_2 = (([], -)\|\cdots\|([], -), \mathcal{R}_1, \mathcal{R}_2, (\varnothing, \Psi_2 \cup \Psi_2'), ((\Psi_1 \cup \Psi_1') \setminus \mathrm{P}_0, S_0)$. From Lemma 53, we have that $\rho_0 \Downarrow h$ iff $\rho_1 \curlywedge^{\epsilon}_{\Pi_0} \rho_2 \Downarrow h$, for all h.

Pick a history h. For the forward direction of the claim, $\rho_0 \Downarrow h$ implies $\rho_1 \curlywedge^{\epsilon}_{\Pi_0} \rho_2 \Downarrow h$ which, from Lemma 52, implies $\rho_1 \otimes^{\epsilon}_{\Pi_0} \rho_2 \Downarrow h$. We now use Lemma 54 to obtain h_1, h_2, σ such that $\rho_i \Downarrow h_i$ and $h = h_1 \curlywedge^{\sigma}_{\Pi_0,\mathrm{P}_0} h_2$. Conversely, suppose that $h_i \in [\![L_i]\!]_N$ and $h = h_1 \curlywedge^{\sigma}_{\Pi_0,\mathrm{P}_0} h_2$. WLOG assume that $(\mathsf{Meths}(h_1) \cup \mathsf{Meths}(h_2)) \cap (\mathsf{dom}(\mathcal{R}_1) \cup \mathsf{dom}(\mathcal{R}_2)) \subseteq \Pi_0$ (or we appropriately alpha-covert $\mathcal{R}_1$ and $\mathcal{R}_2$). Then, $\rho_i \Downarrow h_i$, for $i = 1, 2$, and therefore $\rho_1 \otimes^{\epsilon}_{\Pi_0} \rho_2 \Downarrow h$ by Lemma 54. By Lemma 52 we have that $\rho_1 \curlywedge^{\epsilon}_{\Pi_0} \rho_2 \Downarrow h$, which in turn implies that $\rho_0 \Downarrow h$, i.e. $h \in [\![L_1 \cup L_2]\!]_N$. □

Appendix E. Composition congruence

Theorem 55. *If $L_1 \lhd L_2$ then, for suitably typed L accessing disjoint part of the store than L_1 and L_2, we have $L \cup L_1 \lhd L \cup L_2$.*

Proof. Assume $L_1 \lhd L_2$ and suppose $h_1 \in [\![L \cup L_1]\!]$. By Theorem 50, $h_1 = h' \curlywedge^{\sigma}_{\Pi,\mathrm{P}} h'_1$, where $h' \in [\![L]\!]$ and $h'_1 \in [\![L_1]\!]$. Because $L_1 \lhd L_2$, there exists $h'_2 \in [\![L_2]\!]$ such that $h'_1 \lhd h'_2$, i.e. $h'_1 \lhd^*_{PO} h'_2$. Note that some of the rearrangements necessary to transform h'_1 into h'_2 may concern actions shared by h'_1 and h'; their polarity will then be different in h'. Let h'' be obtained by applying such rearrangements to h'. We claim that $h' \lhd^*_{OP} h''$. Indeed, suppose that $(t', x')(t, x)_P$ are consecutive in h'_1, but swapped in order to obtain h'_2, and $(t, x)_P$ appears in h' as $(t, x)_O$. Now, the move (t', x') either appears in h_1, or it appears in h' and gets hidden in h_1. In every case, let s contain the moves of h' that are after (t', x') in the composition to h_1, and before $(t, x)_O$. We have that $s(t, x)_O$ is a subsequence of h' and $h' \lhd^*_{OP} h''$ holds just if s contains no moves from t. But, if s contained moves from t then the rightmost one such would be some $(t, y)_P$. Moreover, in the composition towards h_1, the move would be scheduled with 1. The latter would break the conditions for trace composition as, at that point, the corresponding subsequence of h'_1 has as leftmost move in t the P-move $(t, x)_P$. We can show similarly that $h' \lhd^*_{OP} h''$ holds in the case that the permutation in h'_1 is on consecutive moves $(t, x)_O(t', x')$. Finally, the rearrangements in h'_1 that do not affect moves shared with h' can be treated in a simpler way: e.g. in the case of $(t', x')(t, x)_P$ consecutive in h'_1 and swapped in h'_2, if $(t, x)_P$ does not appear in h' then we can check that h' cannot contain any t-moves between (t', x') and (t, x) as the conditions for trace composition impose that only O is expected to play in that part of h' (and any t-move would swap this polarity).
Now, since $h' \in [\![L]\!]$, Lemma 34 implies $h'' \in [\![L]\!]$. Take h_2 to be $h'' \curlywedge^{\sigma'}_{\Pi,\mathrm{P}} h'_2$, where σ' is obtained from σ following these move rearrangements. We then have $h_2 \in [\![L \cup L_2]\!]$. Moreover, $h_1 \lhd h_2$ thanks to $h'_1 \lhd h'_2$. Hence, $h_2 \in [\![L \cup L_2]\!]$ and $h_1 \lhd h_2$. Thus, $L \cup L_1 \lhd L \cup L_2$. □

We next examine the behaviour of $\lhd_{\text{enc}}$ with respect to library composition. In contrast to general linearisability, we need to restrict composition for it to be compatible with encapsulation.

Remark 56. The general case of union does not conform with encapsulation in the sense that encapsulated testing of $L \cup L_i$ ($i = 1, 2$) according to Def. 31 may subject L_i to unencapsulated testing. For example, because method names of L and L_i are allowed to overlap, methods in L may call public methods from L_i as well as implementing abstract methods from L_i. This amounts to L playing the role of both $\mathcal{K}$ and $\mathcal{L}$, which in addition can communicate with each other, as both are inside L.

Even if we make L and L_i non-interacting (i.e. without common abstract/public methods), if higher-order parameters are still involved, the encapsulated tests of $L \cup L_i$

can violate the encapsulation hypothesis for L_i. For instance, consider the methods $m_2, m_1', m_2' \in \mathsf{Meths}_{\mathsf{unit},\mathsf{unit}}$ and $m_1 \in \mathsf{Meths}_{(\mathsf{unit}\to\mathsf{unit}),\mathsf{unit}}$, and libraries $L_1, L_2 : \{m_1\} \to \{m_2\}$ and $L : \{m_1'\} \to \{m_2'\}$, as well as the unions $L \cup L_i : \{m_1, m_2\} \to \{m_1', m_2'\}$. A possible trace in $[\![L \cup L_i]\!]_{\mathrm{enc}}$ is this one:

$$\begin{aligned} h_i = {} & (1, \mathsf{call}\, m_2())_{O\mathcal{K}}\, (1, \mathsf{call}\, m_1(v))_{P\mathcal{L}}\, (1, \mathsf{ret}\, m_1())_{O\mathcal{L}} \\ & (1, \mathsf{ret}\, m_2())_{P\mathcal{K}}\, (1, \mathsf{call}\, m_2'())_{O\mathcal{K}}\, (1, \mathsf{call}\, m_1'())_{P\mathcal{L}}\, (1, \mathsf{call}\, v())_{O\mathcal{L}} \end{aligned}$$

which decomposes as $h_i = h' \curlywedge^{\sigma}_{\Pi,\varnothing} h_i'$, with $\Pi = \{m_1, m_2, m_1', m_2'\}$, $\sigma = 2222112$, $h' = (1, \mathsf{call}\, m_2'())_{O\mathcal{K}}\, (1, \mathsf{call}\, m_1'())_{P\mathcal{L}}$ and:

$$h_i' = (1, \mathsf{call}\, m_2())_{O\mathcal{K}}\, (1, \mathsf{call}\, m_1(v))_{P\mathcal{L}}\, (1, \mathsf{ret}\, m_1())_{O\mathcal{L}}\, (1, \mathsf{ret}\, m_2())_{P\mathcal{K}}\, (1, \mathsf{call}\, v())_{O\mathcal{L}}$$

We now see that $h_i' \notin [\![L_i]\!]_{\mathrm{enc}}$ as in the last move O is changing component from $\mathcal{K}$ to $\mathcal{L}$.

We therefore look at compositionality for two specific cases: encapsulated sequencing (e.g. of $L : \Psi \to \Psi'$ with $L' : \Psi' \to \Psi''$) and disjoint union for first-order methods. Given $L : \Psi_1 \to \Psi_2$ and $L' : \Psi_1' \to \Psi_2'$, we define their *disjoint union* $L \uplus L' = L \cup L' : (\Psi_1 \cup \Psi_1') \to (\Psi_2 \cup \Psi_2')$ under the assumption that $(\Psi_1 \cup \Psi_2) \cap (\Psi_1' \cup \Psi_2') = \varnothing$.

Theorem 57. *Let $L_1, L_2 : \Psi_1 \to \Psi_2$ and $L : \Psi_1' \to \Psi_2'$. If $L_1 \triangleleft_{\mathrm{enc}} L_2$ then:*

- *assuming $\Psi_2' = \Psi_1$, we have $L\,;L_1 \triangleleft_{\mathrm{enc}} L\,;L_2$ and $L_1\,;L \triangleleft_{\mathrm{enc}} L_2\,;L$;*
- *if $\Psi_1, \Psi_2, \Psi_1', \Psi_2'$ are first-order then $L \uplus L_1 \triangleleft_{\mathrm{enc}} L \uplus L_2$.*

Proof. Let us consider the first sequencing case (the second one is dual), and assume that $L_1, L_2 : \Psi \to \Psi'$ and $L : \Psi'' \to \Psi$. Assume $L_1 \triangleleft_{\mathrm{enc}} L_2$ and suppose $h_1 \in [\![L\,;L_1]\!]_{\mathrm{enc}}$. By Theorem 50, $h_1 = h' \curlywedge^{\sigma}_{\Pi,\mathrm{P}} h_1'$, where $h' \in [\![L]\!]$, $h_1' \in [\![L_1]\!]$ and method calls from Ψ are always scheduled with 0. The fact that O cannot switch between $\mathcal{L}/\mathcal{K}$ components in (threads of) h_1 implies that the same holds for h', h_1', hence $h' \in [\![L]\!]_{\mathrm{enc}}$ and $h_1' \in [\![L_1]\!]_{\mathrm{enc}}$. Because $L_1 \triangleleft_{\mathrm{enc}} L_2$, there exists $h_2' \in [\![L_2]\!]_{\mathrm{enc}}$ such that $h_1' \triangleleft h_2'$, i.e. $h_1' (\triangleleft_{PO} \cup \diamond)^* h_2'$. As before, some of the rearrangements necessary to transform h_1' into h_2' may concern actions shared by h_1' and h'; we need to check that these can lead to compatible $h'' \in [\![L]\!]_{\mathrm{enc}}$. Let h'' be obtained by applying such rearrangements to h'. We claim that $h' \triangleleft_{OP}^* h''$. The transpositions covered by $\triangleleft_{PO}$ are treated as in Lemma 55. Suppose now that $(t', x')_{P\mathcal{K}} (t, x)_{O\mathcal{L}}$ are consecutive in h_1' but swapped in order to obtain h_2', and $(t, x)_{O\mathcal{L}}$ appears in h' as $(t, x)_{P\mathcal{K}}$. Now, the move (t', x') cannot appear in h' as it is in L_1's $\mathcal{K}$-component (L is the $\mathcal{L}$-component of L_1). Let s contain the moves of h' that are after (t', x') in the composition to h_1, and before $(t, x)_{P\mathcal{K}}$. We claim that s contains no moves from t, so h' can be directly composed with h_2' as far as this transposition is concerned. Indeed, if s contained moves from t then, taking into account the encapsulation conditions, the leftmost one such would be some $(t, y)_{O\mathcal{K}}$. But the $\mathcal{K}$-component of L is L_1, which contradicts the fact that the moves we consider are consecutive in h_1'. Hence, taking h_2 to be $h'' \curlywedge^{\sigma'}_{\Pi,\mathrm{P}} h_2'$, where σ' is obtained from σ following the $\triangleleft_{PO}$ move rearrangements, we have $h_2 \in [\![L\,;L_2]\!]_{\mathrm{enc}}$ and $h_1 \triangleleft_{\mathrm{enc}} h_2$. Thus, $L\,;L_1 \triangleleft_{\mathrm{enc}} L\,;L_2$.

The case of $L \uplus L_1 \triangleleft_{\mathrm{enc}} L \uplus L_2$ is treated in a similar fashion. In this case, because of disjointness, the moves transposed in h_1' do not have any counterparts in h'. Again, we consider consecutive moves $(t', x')_{P\mathcal{K}} (t, x)_{O\mathcal{L}}$ in h_1' that are swapped in order to obtain h_2'. Let s contain the moves of h' that are after (t', x') in the composition to h_1, and before (t, x). As Ψ_1, Ψ_1' is first-order, $(t, x)_{O\mathcal{L}}$ must be a return move and the t-move preceding it in h_1 must be the corresponding call. The latter is a move in h_1', which therefore implies that there can be no moves from t in s. Similarly for the other transposition case. $\square$