CoLoSL: Concurrent Local Subjective Logic

Azalea Raad, Jules Villard, Philippa Gardner

January 9, 2015

Contents

1. Introduction

A key difficulty in verifying properties of shared-memory concurrent programs is to be able to reason compositionally about each thread in isolation, even though in reality the correctness of the whole system is the collaborative result of intricately intertwined actions of the threads. Such compositional reasoning is essential for verifying large concurrent systems, library code and more generally incomplete programs, and for replicating a programmer's intuition about why their implementations are correct.

Rely-guarantee (RG) reasoning [12], is a well-known technique for verifying shared-memory concurrent programs. In this method, each thread specifies its expectations (the rely condition or R) of the transitions made by the environment as well as the transitions made by the thread itself (the guarantee condition or G) where $R; G$ constitute the overall *interference*. However, in practice formulating the rely and guarantee conditions is difficult: the *entire* program state is treated as a shared resource (accessible by all threads) where rely and guarantee conditions *globally* specify the behaviour of threads over the whole shared state and need to be checked throughout the execution of the thread. We proceed with a brief overview of the shortcomings of RG reasoning and how the existing approaches tackle *some* of these limitations. The global nature of RG reasoning limits its compositionality and practicality in the following ways:

1. Even when parts of the state are owned by a single thread, they are exposed to all other threads in the $R; G$ conditions. Simply put, the boundary between private (thread-local) and shared resources is blurred.

2. Since the shared resources are globally known, sharing of *dynamically* allocated resources is difficult. That is, *extending* the shared state is not easy.

3. When parts of the shared state are accessible by only a subset of threads, it is not possible to hide *either* the resources *or* their associated

interference ($R;G$ conditions) from the unconcerned threads. In short, reasoning *locally* about threads with *disjoint* footprints is not possible.

4. Similarly, when different threads access different but *overlapping* parts of the shared state, it is not possible to hide *either* the resources *or* their associated interference from the unconcerned threads. In brief, reasoning *locally* about threads with *overlapping* footprints is not possible. As we will demonstrate, this issue is particularly pertinent when reasoning about concurrent operations on data structures with *unspecified* sharing such as graphs.

5. When describing the specification of a program module, the $R;G$ conditions need to reflect the entire shared state even when the module accesses only parts of it. This limits the *modularity* of verification since the module specification becomes context-dependent and it may not always be reusable.

6. Since the $R;G$ conditions are defined statically and cannot evolve, in order to temporarily disable/enable certain operations by certain threads (e.g. allowing a lock to be released only by the thread who has acquired it) one must appeal to complex (unscalable) techniques such as auxiliary states.

7. As a program executes, its footprint grows/shrinks in tandem with the resources it accesses. It is thus valuable for the reasoning to mimic the programmer's intuition by reflecting the changes in the footprint. This calls for appropriate (de)composition of the shared state as well as its associated interference which cannot be achieved with a global view of the shared state.

Recent work on RGSep [18] has combined the compositionality of separation logic [11, 14] with the concurrent techniques of RG reasoning. In RGSep reasoning, the program state is split into private (thread-local) and shared parts ensuring that the private resources of each thread are untouched by others and the $R;G$ conditions are specified only over the shared state. However, since the shared state itself remains globally specified and is visible to all threads in its entirety, this separation only addresses the first problem outlined above.

Set out to overcome the limitations of both RG and RGSep reasoning, Feng introduced Local Rely-Guarantee (LRG) reasoning in [8]. As in RG and unlike RGSep, in LRG reasoning the program state is treated as a single shared resource accessible by all threads. Moreover, the compositionality

afforded by the separating conjunction of separation logic $*$ is applied to both resources of the shared state *and* the $R;G$ conditions. This way, threads can hide (frame off) irrelevant parts of the shared state and their interference (resolving 1-3, 5 above) allowing for more local reasoning provided that they operate on completely *disjoint* resources. However, when reasoning about data structures with intricate and unspecified sharing (e.g. graphs), since decomposition of overlapping resources is not possible in a disjoint manner, LRG reasoning enforces a global treatment of the shared state, thus betraying its very objective of locality (issue 4; and 5 in the presence of overlapping module specifications). Furthermore, as with RG reasoning, the $R;G$ conditions are specified statically (albeit decomposable); hence temporary (un)blocking of certain actions by certain threads is not easy (issue 6). Finally, while LRG succeeds to capture the programmer's intuition of the program state by dynamically growing/shrinking the footprint when dealing with disjoint resources, it fails to achieve this level of fine-grained locality when dealing with overlapping (entangled) resources (issue 7).

Much like LRG reasoning, the reasoning framework of Concurrent Abstract Predicates (CAP) [6] and its extended variants [17, 3] apply the compositionality of separation logic to concurrent reasoning. In these techniques, the state is split into private (exclusive to each thread) and shared parts where the shared state itself is divided into an arbitrary number of *regions* disjoint from one another. Each region is governed by an *interference* relation I that encompasses both the R and G conditions: the transitions in I are enabled by *capabilities* and a thread holding the relevant capability locally (in its private state) may perform the associated transition. As with LRG, this fine-grained division of the shared state into regions, allows for more local reasoning with the constraint that the the regions are pairwise *disjoint* (resolving 1, 3, 5). Dynamically allocated resources can be shared through creation of new regions (resolving 2). Moreover, since CAP is built on top of *deny-guarantee* reasoning [7] (an extension to rely-guarantee reasoning where *deny* permissions are used to block certain behaviour over a period of time) dynamic manipulation of interference transitions is straightforward (resolving 6). However, since the contents of regions must be disjoint from one another, when tackling data structures with complex and unspecified patterns of sharing that do not lend themselves to a clean decomposition, the entire data structure along with its interference must be confined to a single region, forsaking the notion of locality once again (issue 4; and 5 in case of overlapping module specifications). Furthermore, since the capabilities and interference associated with each region are defined upon its creation and remain unchanged throughout its lifetime, it is often necessary to foresee all

possible extensions to the region (including dynamically allocated resources and their interference). As we will show, this not only limits the locality of reasoning, but also gives way to unnatural specifications that contrast with the program footprint (issue 7).

This paper introduces the program logic of CoLoSL, where threads access one global shared state and are verified with respect to their *subjective views* of this state. Each subjective (personalised) view is an assertion which provides a thread-specific description of *parts* of the shared state. It describes the partial shared resources necessary for the thread to run and the thread-specific interference describing how the thread and the environment may affect these shared resources. Subjective views may arbitrarily overlap with each other, and subtly expand and contract to the natural resources and the interference required by the current thread. This flexibility provides truly compositional reasoning for shared-memory concurrency.

A subjective view $\boxed{P}_I$ comprises an assertion P which describes *parts* of the global shared state; and an interference assertion I which characterises how this partial shared state may be changed by the thread or the environment. Similar to interference assertions of CAP, I declares transitions of the form $[\mathsf{a}] : Q \rightsquigarrow R$, where a thread in possession of the $[\mathsf{a}]$ capability (in its private state) may carry out its transition and update parts of the shared state that satisfy Q to those described by R. Assertions of subjective views must be *stable*; that is, robust with respect to interferences from the environment (as prescribed in I).

We introduce several novel reasoning principles which allow us to expand and contract subjective views. Subjective views can always be duplicated using the Copy principle:

$$\boxed{P}_I \Rightarrow \boxed{P}_I * \boxed{P}_I \qquad \text{(Copy)}$$

In combination with the usual law of parallel composition of separation logic [14], this yields a powerful mechanism to distribute the shared state between several threads:

$$\frac{\{P_1 * P_2\}\ \mathbb{C}_1 \parallel \mathbb{C}_2\ \{Q_1 * Q_2\}}{\{P_1\}\ \mathbb{C}_1\ \{Q_1\} \qquad \{P_2\}\ \mathbb{C}_2\ \{Q_2\}}\ \textsc{Par}$$

The subjective views of each thread will be typically too strong, describing resources that are not being used by the thread. However, as we demonstrate in §2, before weakening the view, it is sometimes useful to strengthen some of the interference transitions to preserve global knowledge. We achieve this using the Shift principle:

$$I \sqsubseteq^P I' \text{ implies } \boxed{P}_I \Rrightarrow \boxed{P}_{I'} \qquad \text{(Shift)}$$

This principle states that an interference assertion I can be exchanged for any other interference assertion I' that has the same projected effect on the subjective state P. When this is the case, we write $I \sqsubseteq^P I'$ and say that the actions have been *shifted*. It can be used to strengthen actions with knowledge of the global shared state arising form the combination of I and P. It can also be used to forget (frame off) actions which are irrelevant to P. This SHIFT principle provides a flexible mechanism for interference manipulation and is in marked contrast with most existing formalisms, where interference is fixed throughout the verification.

With a possibly strengthened interference assertion, we can then frame off parts of the shared state and zoom on resources required by the current thread using the FORGET principle:

$$\boxed{P * Q}_I \Rightarrow \boxed{P}_I \qquad \text{(FORGET)}$$

At this point, the SHIFT principle may be applied again to forget those actions that are no longer relevant to the new subjective view captured by P. However, a stable subjective view may no longer be stable after forgetting parts of the shared state. This is often due to the combined knowledge of P and I being too weak and can be avoided by strengthening I (through SHIFT) prior to forgetting.

These reasoning principles enable us to provide subjective views for the threads which are just right. We can proceed to verify the threads, knowing that their subjective views describe personalised preconditions which only describe the resource relevant to the individual threads. The resulting postconditions naturally describe overlapping parts of the shared state, which are then joined together using the disjoint concurrency rule and the MERGE principle:

$$\boxed{P}_{I_1} * \boxed{Q}_{I_2} \Rightarrow \boxed{P \curlywedge Q}_{I_1 \cup I_2} \qquad \text{(MERGE)}$$

The $P \curlywedge Q$ assertion describes the overlapping of states arising from P and Q, using the overlapping conjunction $\curlywedge$ [10, 9]. The new interference is simply the union of previous interferences. Using the SHIFT principle, we can once again simplify the interference assertion with respect to this new larger subjective view.

Lastly, locally owned resources (described by P) can be shared using the EXTEND principle:

$$P \mathbin{©} I \text{ and } \mathit{fresh}\,(\bar{x}, \mathcal{K}_1 * \mathcal{K}_2) \text{ implies } P \Rrightarrow \exists \bar{x}.\, \mathcal{K}_1 * \boxed{P * \mathcal{K}_2}_I \qquad \text{(EXTEND)}$$

where $\bar{x}$ range over bound logical variables of capability assertions $\mathcal{K}_1$, $\mathcal{K}_2$ and are distinct from unbound variables of P. The side condition $P \circledcirc I$ ensures that the actions of the new interference assertion I are confined to P (more in §4). As we demonstrate in our set example in §5, the main novelty of this rule is that the new actions in I may refer to existing shared resources, unlike CAP where all possible futures of the region must be accounted for upon its creation.

With these reasoning principles, we are able to expand and contract subjective views to provide just the resources required by a thread. In essence, we provide a framing mechanism both on shared resources as well as their interferences even in the presence of overlapping footprints and entangled resources.

We conclude this section by noting that while CoLoSL enables reasoning about *fine-grained* concurrency, the logic of subjective concurrent separation logic (SCSL) [13] employs auxiliary states to reason about *coarse-grained* concurrency. That is, although both techniques share the spirit of *subjectivity*, their applications are completely orthogonal. As the authors note in [13], CAP (and by design CoLoSL) can be employed to reason about their case studies of coarse-grained concurrency. However, SCSL cannot be used to reason about the fine-grained concurrency scenarios that CoLoSL tackles, either when the resources are distributed disjointly amongst threads or when they are intricately entangled.

2. Informal Development

We illustrate our CoLoSL reasoning principles by sketching a proof of a variation of Dijkstra's token ring mutual exclusion algorithm [4].

Consider the program $\mathbb{INC}$ defined in Fig. 2.1, ignoring the assertions. It is written in pseudo-code resembling C with additional constructs for concurrency: atomic sections $\langle \_ \rangle$ which declare that code behaves atomically; and parallel composition $\_ \parallel \_$ which spawns threads then waits until they complete. In our example, after initialisation of the variables to 0, three threads are spawned to increment each variable in a lock-step fashion: $\mathbb{P}_{\mathrm{x}}$ is the first allowed to run its increment operation; then $\mathbb{P}_{\mathrm{y}}$; and finally $\mathbb{P}_{\mathrm{z}}$. This process repeats until $\mathrm{x} = \mathrm{y} = \mathrm{z} = 10$. This example code is interesting because the threads are intricately intertwined. In the case of the $\mathbb{P}_{\mathrm{y}}$ thread, the programmer knows that the code depends on the values of variables x and y and that it can increment y as long as its value is less than that of x. However, he also knows a much more complex behaviour of the thread: given the initial setting where all variables have value 0, then the thread can only increase the value of y by 1 if x is one more than y, and the environment can only increase x by one if x and z (and in fact y) have the same value. Finally, the programmer knows that at the end all the variables will have value 10.

CoLoSL can simply specify this complex behaviour of the resources associated with thread $\mathbb{P}_{\mathrm{y}}$. Consider the CoLoSL assertions accompanying $\mathbb{INC}$. After initialisation, line 3 of $\mathbb{INC}$ provides a standard assertion from separation logic [11] with variables as resource [1]. The assertion declares that the variable cells addressed by x, y and z have the same value 0. This variable resource in the thread-local state is fully owned by the thread. Using the Extend principle, the thread is able to give up this local resource and transfer it to the global shared state. For example, line 4 demonstrates the creation of a subjective view $\boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0 * \mathrm{z} \mapsto 0}_I$, where part of the underlying global shared state now contains the three variable cells and the interference relation I declares how this part of the shared state can change.

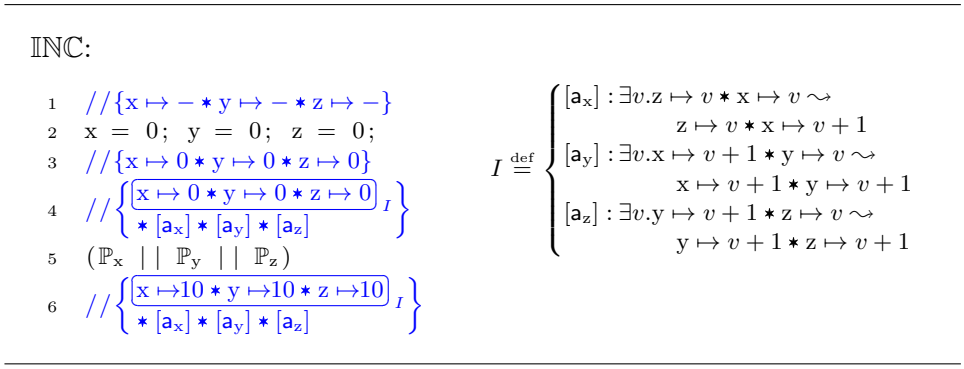

Figure 2.1: The concurrent increment program together with a CoLoSL proof sketch. Lines starting with // contain formulas that describe the local state and the subjective shared state at the relevant program point. The codes of $\mathbb{P}_x$, $\mathbb{P}_y$ and $\mathbb{P}_z$ programs and their proof sketches are provided in Fig. 2.2.

For example, the action

$$[a_y] : \exists v.\ x \mapsto v + 1 * y \mapsto v \rightsquigarrow x \mapsto v + 1 * y \mapsto v + 1$$

can increment y under the condition that the value of x is one more than y. This update is only possible when the local state of a thread has the capability $[a_y]$. For this particular example, the assertion in line 4 simply has all the capabilities $[a_x] * [a_y] * [a_z]$ of the actions of I in the local state; in general, capabilities can be buried inside boxes, only to emerge as a consequence of an action (see §5).

Using the COPY principle for subjective views and the disjoint concurrency rule, we obtain a precondition for thread $\mathbb{P}_y$:

$$\boxed{x \mapsto 0 * y \mapsto 0 * z \mapsto 0}_I * [a_y]$$

However, this precondition is more complicated than we need. Intuitively, the specification of each thread should only use the variable resource relevant to that thread, and need only consider actions that affect that resource. In this example, the extraneous piece of state is the variable cell z. This additional resource might seem an acceptable price to pay, but straightforward generalisations to n participants yields extra state of $n - 2$ variable cells with their associated interferences which are of no interest to the particular thread. Fundamentally, for large systems, the burden of carrying the whole shared state around to analyse all threads, can lead to intractable proofs.

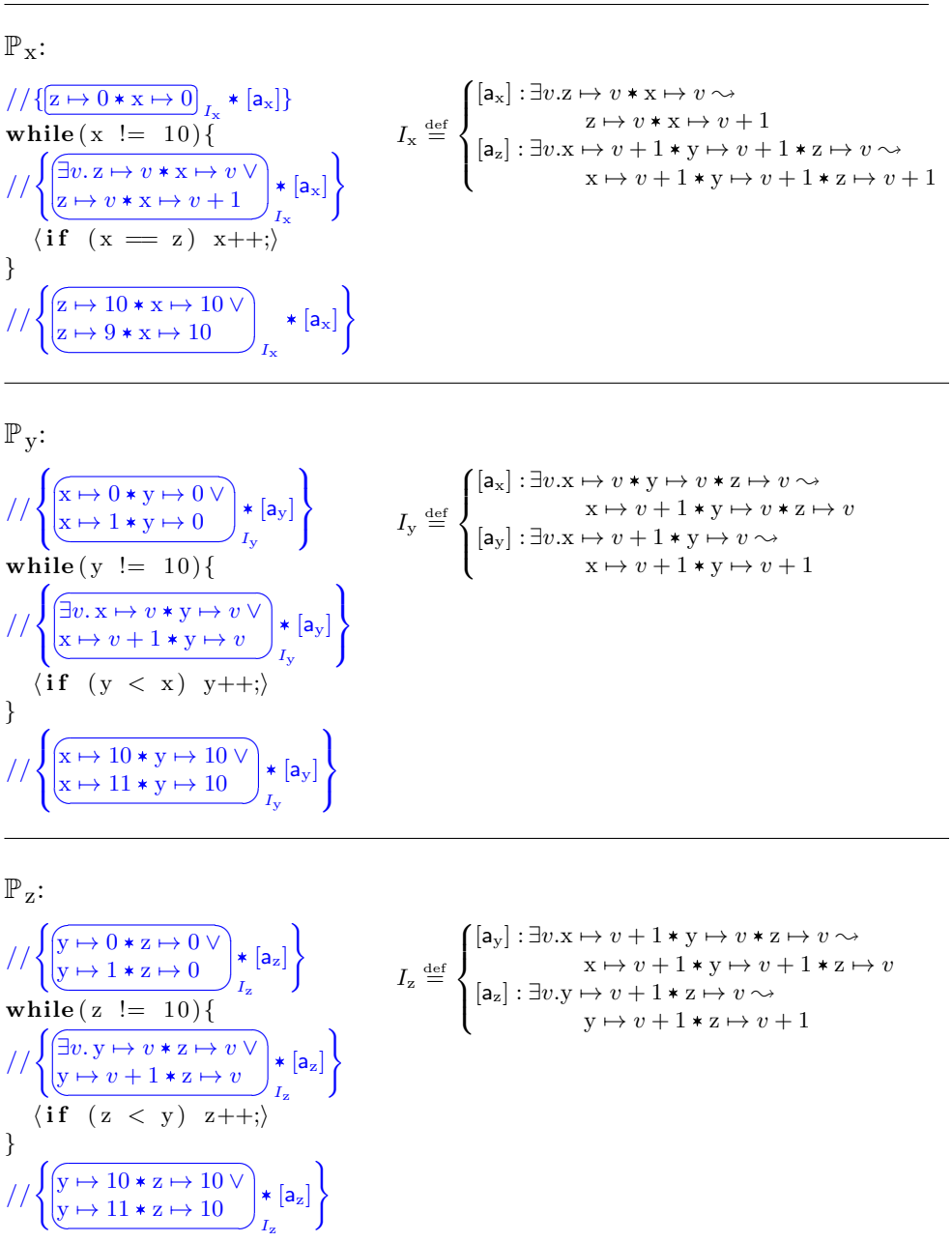

Figure 2.2: The $\mathbb{P}_x$, $\mathbb{P}_y$ and $\mathbb{P}_z$ programs and their CoLoSL proof sketches.

As a first try at simplifying the precondition, consider the following implication using the FORGET and SHIFT principles given in the introduction:

$$
\begin{array}{rl}
 & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0 * \mathrm{z} \mapsto 0}_{I} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(FORGET)}}{\Rightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0}_{I} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(SHIFT)}}{\Rrightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0}_{I \setminus \mathsf{a}_{\mathrm{z}}} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(stabilise)}}{\Rightarrow} & \boxed{\exists v, v'.(\mathrm{x} \mapsto v * \mathrm{y} \mapsto v') \wedge v \geq v'}_{I \setminus \mathsf{a}_{\mathrm{z}}} * [\mathsf{a}_{\mathrm{y}}]
\end{array}
$$

The thread $\mathbb{P}_{\mathrm{y}}$ does not modify z and we can thus forget the variable assertion $\mathrm{z} \mapsto 0$. The variable cell z is no longer visible to $\mathbb{P}_{\mathrm{y}}$ and the action of $[\mathsf{a}_{\mathrm{z}}]$ does not affect the resources described by the assertion of the subjective view neither in its current state nor at any point during its lifetime. That is, the current subjective view is unaffected by this action, and after undergoing any number of actions from I, the resulting subjective view remains unaffected by it. Using the SHIFT principle, we can therefore forget the action of $[\mathsf{a}_{\mathrm{z}}]$.

Finally, we stabilise the resulting subjective view such that it is invariant under all possible actions by the environment. Since we no longer know the value of z, after stabilising against the action of $[\mathsf{a}_{\mathrm{x}}]$, the resulting assertion is too weak. Intuitively, we know that x can only be incremented when its value is equal to z and y. However, this is not reflected in he action of $[\mathsf{a}_{\mathrm{x}}]$; since we have forgotten z, there is nothing to constrain the increment on x. Hence, we can only stabilise in a general way as given, losing information about how the values of x and y are connected together through z. It is however possible to give a stronger specification, as follows:

$$
\begin{array}{rl}
 & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0 * \mathrm{z} \mapsto 0}_{I} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(SHIFT)}}{\Rrightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0 * \mathrm{z} \mapsto 0}_{I'_{\mathrm{y}}} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(FORGET)}}{\Rightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0}_{I'_{\mathrm{y}}} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(SHIFT)}}{\Rrightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0}_{I_{\mathrm{y}}} * [\mathsf{a}_{\mathrm{y}}] \\
\overset{\text{(stabilise)}}{\Rightarrow} & \boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0 \vee \mathrm{x} \mapsto 1 * \mathrm{y} \mapsto 0}_{I_{\mathrm{y}}} * [\mathsf{a}_{\mathrm{y}}]
\end{array}
$$

This implication involves subtle interaction between the assertion and the interference relation of the subjective view. Consider the action of $[\mathsf{a}_{\mathrm{x}}]$ in I and the initial state with value 0 in all the cells. This action can be replaced

by:

$$[\mathsf{a}_{\mathrm{x}}] : \exists v.\ \mathrm{x} \mapsto v * \mathrm{y} \mapsto v * \mathrm{z} \mapsto v \rightsquigarrow \mathrm{x} \mapsto v + 1 * \mathrm{y} \mapsto v * \mathrm{z} \mapsto v$$

This is possible because, as the programmer knows, whenever x and z have the same value then y also has the same value which, under these conditions, is not changed by the actions in I. This amended action reflects stronger knowledge about when x can be incremented and how its value is related to y and z. Let I'_{y} be I_{y} with action of $[\mathsf{a}_{\mathrm{x}}]$ rewritten as above. As we justify in § 4, by using the judgement $I \sqsubseteq^{\mathrm{x}\mapsto 0 * \mathrm{y}\mapsto 0 * \mathrm{z}\mapsto 0} I'_{\mathrm{y}}$, we can use the SHIFT principle to replace I_{y} by I'_{y}. Using FORGET, it is now safe to lose the z assertion to obtain the subjective view $\boxed{\mathrm{x} \mapsto 0 * \mathrm{y} \mapsto 0}_{I'_{\mathrm{y}}}$, as the new action of $[\mathsf{a}_{\mathrm{x}}]$ in I'_{y} retains enough information about how x, y and z are related. Since the action of $[\mathsf{a}_{\mathrm{z}}]$ only affects the z cell, which has been forgotten, leaving the cells x and y unaltered, we can use the SHIFT principle again to change the interference relation to $I_{\mathrm{y}} = I'_{\mathrm{y}} \backslash \mathsf{a}_{\mathrm{z}}$. The interference relation is now as simple as it can get, whilst retaining enough information about the connection between x, y and z. Finally, we stabilise the subjective view with respect to I_{y} and obtain our final precondition of $\mathbb{P}_{\mathrm{y}}$.

There is one more subtlety to mention about this precondition. The thread has the capability $[\mathsf{a}_{\mathrm{y}}]$ to perform its action which only requires the resources described by the subjective view. However, the action of $[\mathsf{a}_{\mathrm{x}}]$ depends on z, which is no longer a part of the subjective view of the thread. Since another thread in the environment may own the $[\mathsf{a}_{\mathrm{x}}]$ capability, it may perform its action whenever its subjective view is *compatible* with the pre-condition of the action. When that is the case, the piece of the state corresponding to the overlap between the state and the precondition of the action is removed, and the entire postcondition of the action is added in its place. This is due to the fact that the subjective view describes the thread's partial knowledge about the shared state, while the environment may have some additional knowledge to what the thread knows. In this case, while the thread does not have the capability to do the action of $[\mathsf{a}_{\mathrm{x}}]$, the environment might.

The proof of the specification of the thread $\mathbb{P}_{\mathrm{y}}$ is now relatively straightforward. By inspection, the invariant of the while loop is stable with respect to I_{y}. The atomic section allows safe manipulation of the contents of the subjective view. The final postcondition of $\mathbb{P}_{\mathrm{y}}$ follows from the invariant and the boolean expression of the while. We join up the postconditions of the threads using the MERGE principle. Since $\vee$ distributes over $\curlyvee$, the subjective view simplifies to $\boxed{\mathrm{x} \mapsto 10 * \mathrm{y} \mapsto 10 * \mathrm{z} \mapsto 10}_{I_{\mathrm{x}} \cup I_{\mathrm{y}} \cup I_{\mathrm{z}}}$. Finally,

since $I_{\mathrm{x}} \cup I_{\mathrm{y}} \cup I_{\mathrm{z}} \sqsubseteq^{\mathrm{x}\mapsto 10 * \mathrm{y}\mapsto 10 * \mathrm{z}\mapsto 10} I$, by the SHIFT principle, we get the postcondition of $\mathbb{INC}$.

This concludes our CoLoSL proof of $\mathbb{INC}$. Our expansion and contraction of subjective views, in particular with shifting of interference assertions in key places, enables us to confine the specification and verification of each thread to just the resources they need. This is not possible in existing frameworks.

Further CoLoSL reasoning We complete our semi-formal overview with a brief description of features of CoLoSL not showcased by the proof above. First, we note that an action $P \rightsquigarrow Q$ may in general perform three types of actions: (1) *modify* resources: P and Q contain the same amount of resource with different values, (e.g. action of $[\mathsf{a}_{\mathrm{x}}]$ in I); (2) *remove* resources from the shared state and transfer them into the local state of the thread performing the action, (e.g. when $P = Q * R$ for some resource R to be removed); (3) conversely, *add* resources from the local state of the thread into the shared state, (e.g. when $Q = P * R$). We demonstrate the latter two behaviours through our examples in §5. These three behaviours are not mutually exclusive and an action may exhibit any combination of them. Unlike CAP [6] and as in iCAP [17], we do not provide an explicit *unsharing* mechanism to claim shared resources and render them thread-local. Instead, this behaviour can be simply encoded as an additional action with the second behaviour described above.

3. CoLoSL Model and Assertions

We formally describe the underlying model of CoLoSL; we present the various ingredients necessary for defining the CoLoSL *worlds*: the building blocks of CoLoSL that track the resources held by each thread, the shared resources accessible to all threads, as well as the ways in which the shared resources may be manipulated by each thread.

We then proceed with the *assertions* of CoLoSL and provide their semantics by relating them to sets of worlds. Finally, we establish the validity of Copy, Forget and Merge principles introduced in the preceding chapters by establishing their truth for all possible worlds and interpretations.

3.1 Worlds

Overview A *world* is a triple $(l, g, \mathcal{I})$ that is *well-formed* where l and g are *logical states* and $\mathcal{I}$ is an *action model*; let us explain the role of each component informally. The *local logical state*, or simply local state, l represents the locally owned resources of a thread. The *shared logical state*, or shared state, g represents the *entire* (global) shared state, accessible to all threads, subject to interferences as described by the action models.

An action model is a partial function from *capabilities* to sets of *actions*. An action is a triple (p, q, c) of logical states where p and q are the *pre* and *post-states* of the action, respectively, and c is the action *condition*. That is, c acts as a mere catalyst for the action: it has to be present for the action to take effect, but is left unchanged by the action. The *action model* $\mathcal{I}$ corresponds directly to the (semantic interpretation of) an interference assertion I. Although worlds do not put further constraints on the relationship between $\mathcal{I}$ and g, they are linked more tightly in the semantics of assertions (§3.2).

Finally, the composition of two worlds will be defined whenever their local states are disjoint and they agree on all other two components, hence

have identical knowledge of the shared state and possible interferences.

We proceed by defining logical states, which are CoLoSL's notion of *resource*, in the standard separation logic sense. Logical states have two components: one describes machine states (*e.g.* stacks and heaps); the other represents *capabilities*. The latter are inspired by the capabilities in deny-guarantee reasoning [7]: a thread in possession of a given capability is allowed to perform the associated actions (as prescribed by the *action model* components of each world, defined below), while any capability *not* owned by a thread means that the environment can perform the action.

CoLoSL is parametric in the choice of the separation algebra representing the machine states and capabilities. This allows for suitable *instantiation* of CoLoSL depending on the programs being verified. For instance, in the token ring example of §2 the separation algebra of machine states is a standard variable stack; while capabilities are captured as a set of tokens. However, as we demonstrate in the examples of §5, our programs often call for a more complex model of machine states and capabilities. For instance, we may need our capabilities to be fractionally owned, where ownership of a *fraction* of a capability grants the right to perform the action to both the thread and the environment, while a fully-owned capability by the thread *denies* the right to the environment to perform the associated action.

In general, the separation algebra of machine states and capabilities can be instantiated with *any* separation algebra (*i.e.* a cancellative, partial commutative monoid [2]) that satisfies the *cross-split* property. This is formalised in the following parametrisations.

Parameter 1 (Machine states separation algebra)**.** Let $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$ be any separation algebra with the cross-split property, representing machine states where the elements of $\mathbb{M}$ are ranged over by $m, m_1, \cdots, m_n$.

Parameter 2 (Capability Separation Algebra)**.** Let $(\mathbb{K}, \bullet_{\mathbb{K}}, \mathbf{0}_{\mathbb{K}})$ be any separation algebra with the cross-split property, representing capability resources where the elements of $\mathbb{K}$ are ranged over by $\kappa, \kappa_1, \cdots \kappa_n$.

We can now formalise the notion of *logical states*. As discussed above, a logical state is a pair comprising a machine state and a capability resource.

Definition 1 (Logical states)**.** Given the separation algebra of machine states, $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$, and the separation algebra of capabilities, $(\mathbb{K}, \bullet_{\mathbb{K}}, \mathbf{0}_{\mathbb{K}})$, a *logical state* is a pair (m, κ), consisting of a machine state $m \in \mathbb{M}$ and a capability $\kappa \in \mathbb{K}$.

$$\mathsf{LState} \stackrel{\text{def}}{=} \mathbb{M} \times \mathbb{K}$$

We write $l, l_1, \cdots, l_n$ to range over either arbitrary logical states or those representing the local logical state. Similarly, we write $g, g_1, \cdots, g_n$ to range over logical states when representing the shared (global) state. We write $\mathbf{0}_{\mathsf{L}}$ for the logical state $(\mathbf{0}_{\mathbb{M}}, \mathbf{0}_{\mathbb{K}})$. Given a logical state l, we write l_{M} and l_{K} for the first and second projections, respectively. The *composition of logical states* $\circ : \mathsf{LState} \times \mathsf{LState} \rightharpoonup \mathsf{LState}$ is defined component-wise:

$$(m, \kappa) \circ (m', \kappa') \stackrel{\text{def}}{=} (m \bullet_{\mathbb{M}} m', \kappa \bullet_{\mathbb{K}} \kappa')$$

The *separation algebra of logical states* is given by $(\mathsf{LState}, \circ, \mathbf{0}_{\mathsf{L}})$.

Oftentimes, we need to compare two logical states $l_1 \leq l_2$ (or their constituents: $\kappa_1 \leq \kappa_2$, $m_1 \leq m_2$) defined when there exists l such that $l \circ l_1 = l_2$. This is captured in the following definition.

Definition 2 (Ordering)**.** Given any separation algebra $(\mathbb{B}, \bullet_{\mathbb{B}}, \mathbf{0}_{\mathbb{B}})$, the *ordering relation*, $\leq : \mathbb{B} \times \mathbb{B}$, is defined as:

$$\leq \stackrel{\text{def}}{=} \{(b_1, b_2) \mid \exists b.\, b_1 \bullet_{\mathbb{B}} b = b_2\}$$

We write $b_1 \leq b_2$ for $(b_1, b_2) \in \leq$. We also write $b_2 - b_1$ to denote the unique (by cancellativity of separation algebras) element in $\mathbb{B}$ such that $b_1 \bullet_{\mathbb{B}} (b_2 - b_1) = b_2$.

In our formalisms we occasionally need to quantify over *compatible* logical states (similarly, compatible machine states or compatible capabilities), *i.e.* those that can be composed together by $\circ$. Additionally, we often describe two logical states as *disjoint*. We formalise these notions below.

Definition 3 (Compatibility)**.** Given any separation algebra $(\mathbb{B}, \bullet_{\mathbb{B}}, \mathbf{0}_{\mathbb{B}})$, the *compatibility relation*, $\sharp : \mathbb{B} \times \mathbb{B}$, is defined as:

$$\sharp \stackrel{\text{def}}{=} \{(b_1, b_2) \mid \exists b.\, b_1 \bullet_{\mathbb{B}} b_2 = b\}$$

We write $b_1 \sharp b_2$ for $(b_1, b_2) \in \sharp$.

Definition 4 (Disjointness)**.** Given any separation algebra $(\mathbb{B}, \bullet_{\mathbb{B}}, \mathbf{0}_{\mathbb{B}})$, the *disjointness relation*, $\perp : \mathbb{B} \times \mathbb{B}$, is defined as:

$$\perp \stackrel{\text{def}}{=} \{(b_1, b_2) \mid b_1 \sharp b_2 \wedge \forall b \in \mathbb{B}.\, b \leq b_1 \wedge b \leq b_2 \implies b = \mathbf{0}_{\mathbb{B}}\}$$

We write $b_1 \perp b_2$ for $(b_1, b_2) \in \perp$.

Observe that for a separation algebra $(\mathbb{B}, \bullet_{\mathbb{B}}, \mathbf{0}_{\mathbb{B}})$ satisfying the disjointness property[1], the definitions of compatibility and disjointness relations coincide.

We now proceed with the next ingredients of a CoLoSL world, namely, action models. Recall from above that an action is simply a pair of logical states describing the pre- and post-states of the action, while an action model describes the set of actions associated with each capability.

Definition 5 (Actions, action models)**.** The set of *actions*, Action, ranged over by $a, a_1, \cdots, a_n$, is as defined below.

$$\mathsf{Action} \stackrel{\text{def}}{=} \mathsf{LState} \times \mathsf{LState} \times \mathsf{LState}$$

The set of *action models*, AMod, is defined as follows.

$$\mathsf{AMod} \stackrel{\text{def}}{=} \mathbb{K} \rightharpoonup \mathcal{P}(\mathsf{Action})$$

We write $\mathcal{I}, \mathcal{I}_1, \cdots, \mathcal{I}_n$ to range over action models; we write $\emptyset$ for an action model with empty domain.

The Effect of Actions Given a world $(l, g, \mathcal{I})$, since g represents the *entire* shared state, as part of the well-formedness condition of worlds we require that the actions in $\mathcal{I}$ are *confined* to g. Let us elaborate on the necessity of the confinement condition.

As threads may unilaterally decide to introduce part of their local states into the shared state at any point (by Extend), confinement ensures that existing actions cannot affect future extensions of the shared state. Similarly, we require that the new actions associated with newly shared state are confined to that extension in the same vein, hence extending the shared state cannot retroactively invalidate the views of other threads. However, as we will demonstrate, confinement does not prohibit *referring* to existing parts of the shared state in the new actions; rather, it only safeguards against *mutation* of the already shared resources through new actions.

Through confinement we ensure that the *effect* of actions in the local and global action models are confined to the shared state. In other words, given an action $a = (p, q, c)$ and a shared state g, whenever $p \circ c$ *agrees* with g then p must be contained in g. Agreement of $p \circ c$ and g merely means that they agree on the resources they have in common. In particular, g only needs to contain p (and not $p \circ c$) for a to take effect. This relaxation is due

[1] $\forall b, b' \in \mathbb{B}.\, b \bullet_{\mathbb{B}} b = b' \implies b = b' = \mathbf{0}_{\mathbb{B}}$

to the fact that other threads may extend the shared state; in particular, the extension may provide the missing resources for $p \circ c$ to be contained in the shared state, thus allowing the extending thread to perform action a. Crucially, however, the part of the shared state mutated by the action, namely p, must be contained in g so that extensions of the shared state need not worry about existing actions interfering with new resources that were never shared beforehand.

The agreement of two logical states (e.g. $p \circ c$ and g above) will be defined using the following notion of *intersection* of logical states. This will enable us to define our confinement condition.

Definition 6 (Intersection)**.** The *intersection* function over logical states, $\sqcap : (\mathsf{LState} \times \mathsf{LState}) \to \mathcal{P}(\mathsf{LState})$, is defined as follows.

$$l_1 \sqcap l_2 \stackrel{\text{def}}{=} \left\{ l \mid \exists l', l'_1, l'_2.\ l_1 = l \circ l'_1 \wedge l_2 = l \circ l'_2 \wedge l \circ l'_1 \circ l'_2 = l' \right\}$$

Observe that when the separation algebra of logical states satisfies the disjointness property, $l_1 \sqcap l_2$ yields at most one element for any l_1 and l_2.

Two logical states l_1 and l_2 then *agree* if their intersection is non-empty, *i.e.* $l_1 \sqcap l_2 \neq \emptyset$. We can now define action confinement.

Definition 7 (Action confinement)**.** An action $a = (p, q, c)$ is *confined* to a logical state g, written $g \copyright a$, if for all r compatible with g ($g \sharp r$):

$$p \circ c \sqcap g \neq \emptyset \Rightarrow p \leq g \wedge p \perp r$$

As discussed, only the action pre-state p, i.e. the part actually mutated by the action has to be contained in g and must be disjoint from all potential extensions (r) of the logical state g. That is, future extensions of g need not account for existing actions interfering with new resources.

Given a shared state g and an action model $\mathcal{I}$, we require that all actions of $\mathcal{I}$ are confined in all possible *futures* of g, *i.e.* all shared states resulting from g after any number of applications of actions in $\mathcal{I}$. For that we define *action application* that describes the effect of an action on a logical state. Moreover, for some of the actions in $\mathcal{I}$, the pre-state may not affect g, *i.e.* its intersection with g may be the empty state $\mathbf{0}_{\mathsf{L}}$. In that case, we find that even though that action is potentially enabled, we do not need to account for it since it leaves g unchanged. We thus introduce the notion of *visible actions* to quantify over those actions that affect (mutate) g.

Definition 8 (Action application)**.** The *application* of an action $a = (p, q, c)$ on a logical state g, written $a[g]$, is defined provided that there exists l such that

$$p \circ c \sqcap g \neq \emptyset \wedge g = p \circ l \wedge q \sharp l$$

When that is the case, we write $a[g]$ for the (uniquely defined) logical state $q \circ l$. We write $\mathit{potential}(a, g)$ to denote that $a[g]$ is defined.

Definition 9 (Visible actions)**.** An action $a = (p, q, c)$ is called *visible in* g, written $\mathit{visible}(a, g)$ when

$$\exists l \in (p \sqcap g) \,.\, l \neq \mathbf{0}_{\mathsf{L}}$$

We are now ready to define our confinement condition on action models. Inspired by Local RG [8], we introduce the concept of locally fenced action models to capture all possible states reachable from the current state via some number of action applications. A set of states $\mathcal{F}$ *fences* an action model if it is invariant under interferences perpetrated by the corresponding actions. An action model is then confined to a logical state l if it can be fenced by a set of states that includes l. In the following we write $rg(f)$ to denote the *range* (or co-domain) of a function f.

Definition 10 (Locally-fenced action model)**.** An action model $\mathcal{I} \in \mathsf{AMod}$ is *locally fenced* by $\mathcal{F} \in \mathcal{P}(\mathsf{LState})$, written $\mathcal{F} \blacktriangleright \mathcal{I}$, iff for all $g \in \mathcal{F}$ and all $a \in rg(\mathcal{I})$,

$$g \copyright a \wedge (\mathit{potential}(a, g) \Rightarrow a[g] \in \mathcal{F})$$

Definition 11 (Action model confinement)**.** An action model $\mathcal{I}$ is *confined* to a logical state l, written $l \copyright \mathcal{I}$, if there exists a fence $\mathcal{F}$ such that $l \in \mathcal{F}$ and $\mathcal{F} \blacktriangleright \mathcal{I}$.

We are almost in a position to define well-formedness of worlds. Since capabilities enable the manipulation of the shared state through their associated actions in the action models, for a world $(l, g, \mathcal{I})$ to be well-formed the capabilities found in the local state l and shared state g must be *contained* in the action model $\mathcal{I}$. That is, *all* capabilities found in the combined state $l \circ g$ must be accounted for in $\mathcal{I}$.

Definition 12 (Capability containment)**.** A capability $\kappa \in \mathbb{K}$, is *contained* in an action model $\mathcal{I} \in \mathsf{AMod}$, written $\kappa \prec \mathcal{I}$ iff

$$\exists K \in \mathcal{P}(\mathbb{K}) \,.\, \kappa = \prod_{\kappa_i \in K}^{\bullet_{\mathbb{K}}} \kappa_i \wedge \forall \kappa_i \in K. \, \exists \kappa' \in \mathsf{dom}(\mathcal{I}) \,.\, \kappa_i \leq \kappa'$$

The above states that κ is contained in $\mathcal{I}$ iff κ is the composition of smaller capabilities $\kappa_i \in K$ where each constituent κ_i is accounted for in the domain of $\mathcal{I}$.

In order to ensure indefinite extension of the shared state and creation of fresh capabilities (through EXTEND principle), as part of the well-formedness condition we require that the domain of action models are well-defined. That is, one can always guarantee the existence of a capability that is fresh (disjoint from the domain of the action model); and that the domain remains well-defined after extension. This is captured by the following definition of well-defined capability sets.

Definition 13 (Well-defined capability subsets)**.** A capability set $K \in \mathcal{P}(\mathbb{K})$ is *well defined*, written $\diamond K$, if and only if there exists a set $\chi \in \mathcal{P}(\mathcal{P}(\mathbb{K}))$ such that $K \in \chi$ and

$$\begin{array}{c} \forall \kappa \in \mathbb{K}.\, \{\kappa\} \in \chi \\ \wedge\, \forall K' \in \chi.\, \exists \kappa \in \mathbb{K}.\, \kappa \perp K' \\ \wedge\, \forall K_1, K_2 \in \chi.\, K_1 \cup K_2 \in \chi \end{array}$$

where we write $\kappa \perp K$ to denote $\forall \kappa' \in K.\, \kappa \perp \kappa'$.

We can now formalise the notion of well-formedness. A world $(l, g, \mathcal{I})$ is well-formed if l and g are compatible, the capabilities found in $l \circ g$ are contained in the action model $\mathcal{I}$,$\mathcal{I}$ is confined to g, and the domain of $\mathcal{I}$ is well-defined.

Definition 14 (Well-formedness)**.** A 4-tuple $(l, g, \mathcal{I})$ is *well-formed*, written $\mathsf{wf}(l, g, \mathcal{I})$, iff

$$(\exists m, \kappa.\, l \circ g = (m, \kappa) \wedge \kappa \prec \mathcal{I}) \wedge g \mathbin{©} \mathcal{I} \wedge \diamond(\mathsf{dom}(\mathcal{I}))$$

Definition 15 (Worlds)**.** The set of *worlds* is defined as

$$\mathsf{World} \stackrel{\text{def}}{=} \{w \in \mathsf{LState} \times \mathsf{LState} \times \mathsf{AMod} \mid \mathsf{wf}(w)\}$$

The *composition* $w \bullet w'$ of two worlds $\bullet : \mathsf{World} \to \mathsf{World} \rightharpoonup \mathsf{World}$, is defined as follows.

$$(l, g, \mathcal{I}) \bullet (l', g', \mathcal{I}') \stackrel{\text{def}}{=} \begin{cases} (l \circ l', g, \mathcal{I}) & \text{if } g = g', \text{ and } \mathcal{I} = \mathcal{I}' \\ & \text{and } \mathsf{wf}((l \circ l', g, \mathcal{I})) \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The set of worlds with composition $\bullet$ forms a separation algebra with multiple units: all well-formed states of the form $(\mathbf{0}_{\mathsf{L}}, g, \mathcal{I})$. Given a world w, we write w_{L} for the first projection.

3.2 Assertions

Our assertions extend standard assertions from separation logic with *subjective views* and *capability assertions*. We assume an infinite set, LVar, of *logical variables* and a set of *logical environments* $\iota \in \mathsf{LEnv} : \mathcal{P}(\mathsf{LVar} \to \mathsf{Val})$ that associate logical variables with their values.

CoLoSL is parametric with respect to the machine states and capability assertions and can be instantiated with any assertion language over machine states $\mathbb{M}$ and capabilities $\mathbb{K}$. This is captured by the following parameters.

Parameter 3 (Machine state assertions)**.** Assume a set of *machine state assertions* MAssn, ranged over by $\mathcal{M}, \mathcal{M}_1, \cdots, \mathcal{M}_n$ and an associated semantics function:

$$(\!|.|\!)^{M}_{(.)} : \mathsf{MAssn} \to \mathsf{LEnv} \to \mathcal{P}(\mathbb{M})$$

Parameter 4 (Capability assertions)**.** Assume a set of capability assertions KAssn ranged over by $\mathcal{K}, \mathcal{K}_1, \cdots, \mathcal{K}_n$ and an associated semantics function:

$$(\!|.|\!)^{K}_{(.)} : \mathsf{KAssn} \to \mathsf{LEnv} \to \mathcal{P}(\mathbb{K})$$

Definition 16 (Assertion syntax)**.** The assertions of CoLoSL are elements of Assn described by the grammar below, where x ranges over logical variables.

$$\begin{aligned} A &::= \mathit{false} \mid \mathsf{emp} \mid \mathcal{M} \mid \mathcal{K} \\ \mathsf{LAssn} \ni p, q &::= A \mid \neg p \mid \exists x.\, p \mid p \vee q \mid p * q \mid p \circledast q \mid p \mathbin{\textcircled{*}} q \mid p \mathrel{-\!\!\textcircled{*}} q \\ \mathsf{Assn} \ni P, Q &::= p \mid \exists x.\, P \mid P \vee Q \mid P * Q \mid P \mathbin{\textcircled{*}} Q \mid \boxed{P}_I \\ \mathsf{IAssn} \ni I &::= \emptyset \mid \{\mathcal{K} : \exists \bar{y}.\, P \rightsquigarrow Q\} \cup I \end{aligned}$$

This syntax follows from standard separation logic, with the exception of subjective views $\boxed{P}_I$. emp is true of the units of $\bullet$. Machine state assertions ($\mathcal{M}$) and capability assertions ($\mathcal{K}$) are interpreted over a world's local state: $\mathcal{M}$ is true of a local state $(m, \mathbf{0}_{\mathbb{K}})$ where m satisfies $\mathcal{M}$; similarly, $\mathcal{K}$ is true of a local state $(\mathbf{0}_{\mathbb{M}}, \kappa)$ where κ satisfies $\mathcal{K}$. $P * Q$ is true of worlds that can be split into two according to $\bullet$ such that one state satisfies P and the other satisfies Q; $P \mathbin{\textcircled{*}} Q$ is the *overlapping conjunction*, true of worlds can be split three-way according to $\bullet$, such that the $\bullet$-composition of the first two worlds satisfies P and the $\bullet$-composition of the last two satisfy Q [16]; classical predicates and connectives have their standard classical meaning. Interference assertions I describe actions enabled by a given capability, in the form of a pre- and post-condition.

A subjective view $\boxed{P}_I$ is true of $(l, g, \mathcal{I})$ when $l = \mathbf{0}_{\mathsf{L}}$ and a subjective view s can be found in the global shared state g, *i.e.* $g = s \circ r$ for some *context* r, such that s satisfies P in the standard separation logic sense, and I and $\mathcal{I}$ *agree* given the decomposition s, r, in the following sense:

1. every action in I is reflected in $\mathcal{I}$;
2. every action in $\mathcal{I}$ that is potentially enabled in g and has a visible effect on s is reflected in I;
3. the above holds after any number of action applications in $\mathcal{I}$ affecting g

These conditions will be captured by the *action model closure* relation $\mathcal{I}{\downarrow}\,(s, r, \langle\!|I|\!\rangle_\iota)$ given by the upcoming Def. 24 (where $\langle\!|I|\!\rangle_\iota$ is the interpretation of I given a logical environment ι).

The semantics of CoLoSL assertions is given by a forcing relation $w, \iota \vDash P$ between a world w, a logical environment $\iota \in \mathsf{LEnv}$, and a formula P. We use two auxiliary forcing relations. The first one $l, \iota \vDash_{\mathsf{SL}} P$ interprets formulas P in the usual separation logic sense over a logical state l (and ignores shared state assertions). The second one $s, \iota \vDash_{g,\mathcal{I}} P$ interprets assertions over a *subjective view* s that is part of the global shared state g, subject to action model $\mathcal{I}$. This third form of satisfaction is needed to deal with nesting of subjective views. We often write $\vDash_\dagger$ as a shorthand for $\vDash_{g,\mathcal{I}}$ when we do not need to refer to the individual components g and $\mathcal{I}$.

Note that this presentation with several forcing relations differs from the usual CAP presentation [6], where formulas are first interpreted over worlds that are not necessarily well-formed, and then cut down to well-formed ones. The CAP presentation strays from separation logic models in some respects; for instance, in CAP, $\ast$ is not the adjoint of $-\!\!\ast$, the "magic wand" connective of separation logic. Although we have omitted this connective from our presentation, its definition in CoLoSL would be standard and satisfy the adjunction with $\ast$.

Definition 17 (Assertion semantics)**.** Given a logical environment $\iota \in \mathsf{LEnv}$, the semantics of CoLoSL assertions is as follows, where $\langle\!|.|\!\rangle_{(.)} : \mathsf{IAssn} \to \mathsf{LEnv} \to \mathsf{AMod}$ denotes the semantics of interference assertions and $\mathcal{I}\downarrow (s, r, \langle\!|I|\!\rangle_\iota)$ will be given in Def. 24.

$$
\begin{array}{lll}
(l, g, \mathcal{I}), \iota \vDash p & \text{iff} & l, \iota \vDash_{\mathsf{SL}} p \\
(l, g, \mathcal{I}), \iota \vDash \boxed{P}_I & \text{iff} & l = \mathbf{0}_{\mathbb{M}} \text{ and } \exists s, r.\, g = s \circ r \text{ and} \\
 & & s, \iota \vDash_{g,\mathcal{I}} P \text{ and } \mathcal{I}{\downarrow}\,(s, r, \langle\!|I|\!\rangle_\iota)
\end{array}
$$

$$
\begin{array}{lll}
w, \iota \vDash \exists x.\, P & \text{iff} & \exists v.\, w, [\iota \mid x : v] \vDash P \\
w, \iota \vDash P \vee Q & \text{iff} & w, \iota \vDash P \text{ or } w, \iota \vDash Q \\
w, \iota \vDash P_1 * P_2 & \text{iff} & \exists w_1, w_2.\, w = w_1 \bullet w_2 \text{ and} \\
 & & w_1, \iota \vDash P_1 \text{ and } w_2, \iota \vDash P_2 \\
w, \iota \vDash P_1 \circledast P_2 & \text{iff} & \exists w', w_1, w_2.\, w = w' \bullet w_1 \bullet w_2 \text{ and} \\
 & & w' \bullet w_1, \iota \vDash P_1 \text{ and } w' \bullet w_2, \iota \vDash P_2
\end{array}
$$

where

$$
\begin{array}{lll}
s, \iota \vDash_{g,\mathcal{J}} p & \text{iff} & s, \iota \vDash_{\mathsf{SL}} p \\
s, \iota \vDash_{g,\mathcal{J}} \boxed{P}_I & \text{iff} & (s, g, \mathcal{J}), \iota \vDash \boxed{P}_I \\
s, \iota \vDash_\dagger \exists x.\, P & \text{iff} & \exists v.\, s, [\iota \mid x : v] \vDash_\dagger P \\
s, \iota \vDash_\dagger P \vee Q & \text{iff} & s, \iota \vDash_\dagger P \text{ or } s, \iota \vDash_\dagger Q \\
s, \iota \vDash_\dagger P_1 * P_2 & \text{iff} & \exists s_1, s_2.\, s = s_1 \circ s_2 \text{ and} \\
 & & s_1, \iota \vDash_\dagger P_1 \text{ and } s_2, \iota \vDash_\dagger P_2 \\
s, \iota \vDash_\dagger P_1 \circledast P_2 & \text{iff} & \exists s', s_1, s_2.\, s = s' \circ s_1 \circ s_2 \text{ and} \\
 & & s' \circ s_1, \iota \vDash_\dagger P_1 \text{ and } s' \circ s_2, \iota \vDash_\dagger P_2
\end{array}
$$

$$
\begin{array}{lll}
l, \iota \vDash_{\mathsf{SL}} \mathit{false} & & \text{never} \\
l, \iota \vDash_{\mathsf{SL}} \mathsf{emp} & \text{iff} & l = \mathbf{0}_{\mathsf{L}} \\
l, \iota \vDash_{\mathsf{SL}} \mathcal{M} & \text{iff} & \exists m.\, l = (m, \mathbf{0}_{\mathbb{K}}) \text{ and } m \in (\!|\mathcal{M}|\!)^M_\iota \\
l, \iota \vDash_{\mathsf{SL}} \mathcal{K} & \text{iff} & \exists \kappa.\, l = (\mathbf{0}_{\mathbb{M}}, \kappa) \text{ and } \kappa \in (\!|\mathcal{K}|\!)^K_\iota \\
l, \iota \vDash_{\mathsf{SL}} \neg p & \text{iff} & l, \iota \nvDash_{\mathsf{SL}} p \\
l, \iota \vDash_{\mathsf{SL}} p \mathrel{-\!\!\circledast} q & \text{iff} & \exists l'.\, l', \iota \vDash_{\mathsf{SL}} p \text{ and } l \sharp l' \\
 & & \text{ implies } l \circ l', \iota \vDash_{\mathsf{SL}} q \\
l, \iota \vDash_{\mathsf{SL}} P_1 * P_2 & \text{iff} & \exists l_1, l_2.\, l = l_1 \circ l_2 \text{ and} \\
 & & l_1, \iota \vDash_{\mathsf{SL}} P_1 \text{ and } l_2, \iota \vDash_{\mathsf{SL}} P_2 \\
l, \iota \vDash_{\mathsf{SL}} P \vee Q & \text{iff} & l, \iota \vDash_{\mathsf{SL}} P \text{ or } l, \iota \vDash_{\mathsf{SL}} Q \\
l, \iota \vDash_{\mathsf{SL}} \exists x.\, P & \text{iff} & \exists v.\, l, [\iota \mid x : v] \vDash_{\mathsf{SL}} P \\
l, \iota \vDash_{\mathsf{SL}} P_1 \circledast P_2 & \text{iff} & \exists l', l_1, l_2.\, l = l' \circ l_1 \circ l_2 \text{ and} \\
 & & l' \circ l_1, \iota \vDash_{\mathsf{SL}} P_1 \text{ and } l' \circ l_2, \iota \vDash_{\mathsf{SL}} P_2
\end{array}
$$

and

$$\langle\!|I|\!\rangle_\iota(\kappa) \stackrel{\text{def}}{=} \left\{ (p,q,c\circ l) \;\middle|\; \begin{array}{l} \mathcal{K} : \exists \vec{y}.\, P \rightsquigarrow Q \in I \wedge \kappa \in (\!|\mathcal{K}|\!)^K_\iota \wedge \\ \exists \vec{v}, \mathfrak{I}. \\ \quad p\circ c, [\iota \mid \vec{y} : \vec{v}] \vDash_{(p\circ c\circ r\circ l),\mathfrak{I}} P \wedge \\ \quad q\circ c, [\iota \mid \vec{y} : \vec{v}] \vDash_{(q\circ c\circ r\circ l),\mathfrak{I}} Q \wedge \\ \quad \forall l'.\, l' \leq p \wedge l' \leq q \implies l' = \mathbf{0}_{\mathsf{L}} \end{array} \right\}$$

Given a logical environment ι, we write $[\![P]\!]_\iota$ for the set of all worlds satisfying P. That is,

$$[\![P]\!]_\iota \stackrel{\text{def}}{=} \{w \mid w, \iota \vDash P\}$$

Although CoLoSL allows for nested subjective views (e.g. $\boxed{P * \boxed{Q}_{I'}}_I$), it is often useful to *flatten* them into equivalent assertions with no nested boxes. We introduce *flat assertions*, $\mathsf{FAssn} \subset \mathsf{Assn}$, to capture a subset of assertions with no nested subjected views and provide a *flattening mechanism* to rewrite CoLoSL assertions as equivalent flat assertions.

Definition 18 (Flattening)**.** The set of *flat assertions* FAssn is defined by the following grammar.

$$\mathsf{FAssn} \ni P, Q ::= p \mid \boxed{p}_I \mid \exists x.\, P \mid P \vee Q \mid P * Q \mid P \mathbin{-\!\!*} Q$$

The *flattening* function $\mathsf{f}(.) : \mathsf{Assn} \to \mathsf{FAssn}$ is defined inductively as follows where x does not appear free in I and $\odot \in \{\vee, *, \mathbin{-\!\!*}\}$.

$$\begin{array}{ll} \mathsf{f}(p) \stackrel{\text{def}}{=} p & \mathsf{f}(P \odot Q) \stackrel{\text{def}}{=} \mathsf{f}(P) \odot \mathsf{f}(Q) \\ \mathsf{f}(\exists x.\, P) \stackrel{\text{def}}{=} \exists x.\, \mathsf{f}(P) & \mathsf{f}\left(\boxed{\boxed{P}_I * Q}_{I'}\right) \stackrel{\text{def}}{=} \mathsf{f}\left(\boxed{P}_I\right) * \mathsf{f}\left(\boxed{Q}_{I'}\right) \\ \mathsf{f}\left(\boxed{p}_I\right) \stackrel{\text{def}}{=} \boxed{p}_I & \mathsf{f}\left(\boxed{\boxed{P}_I \mathbin{-\!\!*} Q}_{I'}\right) \stackrel{\text{def}}{=} \mathsf{f}\left(\boxed{P}_I\right) * \mathsf{f}\left(\boxed{Q}_{I'}\right) \\ \mathsf{f}\left(\boxed{\exists x.\, P}_I\right) \stackrel{\text{def}}{=} \exists x.\, \mathsf{f}\left(\boxed{P}_I\right) & \mathsf{f}\left(\boxed{P \vee Q}_I\right) \stackrel{\text{def}}{=} \mathsf{f}\left(\boxed{P}_I\right) \vee \mathsf{f}\left(\boxed{Q}_I\right) \end{array}$$

Definition 19 (Erasure)**.** The *erasure* of an assertion $\lfloor(.) : \mathsf{Assn} \to \mathsf{LAssn}$ is defined inductively over the structure of assertions as follows with $\odot \in \{\vee, *, \mathbin{-\!\!*}\}$

$$\lfloor A \stackrel{\text{def}}{=} A \qquad \lfloor\left(\boxed{P}_I\right) \stackrel{\text{def}}{=} \mathsf{emp} \qquad \lfloor(\exists x.\, P) \stackrel{\text{def}}{=} \exists x.\, \lfloor P \qquad \lfloor(P \odot Q) \stackrel{\text{def}}{=} \lfloor P \odot \lfloor Q$$

Definition 20 (Gather/merge)**.** Given an assertion $P \stackrel{\text{def}}{=} \exists \bar{x}.\, P' \in \mathsf{Assn}$, where P is in the prenex normal form with no bound variables in P',the

gather, $\bigotimes(.) : \mathsf{Assn} \to \mathsf{LAssn}$, and *merge*, $\bigoplus(.) : \mathsf{Assn} \to \mathsf{LAssn}$, operations are defined as follows:

$$\bigotimes P \stackrel{\text{def}}{=} \exists \bar{x}.\ p * q \qquad \bigoplus P \stackrel{\text{def}}{=} \exists \bar{x}.\ p \circledast q \qquad \text{where } (p, q) = \mathsf{ub}\left(\mathsf{f}\left(P'\right)\right)$$

with the auxiliary function $\mathsf{ub}\,(.) : \mathsf{FAssn} \to (\mathsf{LAssn} \times \mathsf{LAssn})$ defined inductively over the structure of local assertions as follows where $\odot \in \{*, \circledast\}$. In what follows, $\mathsf{ub}\,(P) = (p, p')$ and $\mathsf{ub}\,(Q) = (q, q')$ where applicable.

$$\mathsf{ub}\,(p) \stackrel{\text{def}}{=} (p, \mathsf{emp}) \qquad \mathsf{ub}\left(\boxed{p}_I\right) \stackrel{\text{def}}{=} (\mathsf{emp}, p) \qquad \mathsf{ub}\,(P \odot Q) \stackrel{\text{def}}{=} \left(p \odot q, p' \circledast q'\right)$$

$$\mathsf{ub}\,(P \vee Q) \stackrel{\text{def}}{=} \left(p \vee q, p' \vee q'\right)$$

Definition 21 (Interference entailment)**.** An interference assertion I *entails* interference assertion I', written $I \vdash_{\mathrm{I}} I'$, if

$$\forall \iota \in \mathsf{LEnv}.\ \langle\!| I |\!\rangle_\iota \subseteq \langle\!| I' |\!\rangle_\iota$$

Lemma 1. The following judgements are valid.

$$\frac{}{P \Leftrightarrow \mathsf{f}\,(P)} \qquad \frac{}{P \vdash \downarrow P} \qquad \frac{P \vdash Q}{\boxed{P}_I \vdash \boxed{Q}_I} \qquad \frac{P \circledast Q \vdash \mathit{false}}{\boxed{P}_I * \boxed{Q}_{I'} \vdash \mathit{false}}$$

$$\frac{R \vdash P \quad P \circledast Q \vdash R \circledast Q}{\boxed{P}_I * \boxed{Q}_{I'} \vdash \boxed{R}_I * \boxed{Q}_{I'}} \qquad \frac{I_1 \vdash_{\mathrm{I}} I_2}{I \cup I_1 \vdash_{\mathrm{I}} I \cup I_2}$$

$$\frac{}{\{\mathcal{K} : \exists \bar{y}.\ P \rightsquigarrow Q\} \vdash_{\mathrm{I}} \{\mathcal{K} : \exists \bar{y}.\ \bigoplus P \rightsquigarrow \bigoplus Q\}}$$

$$\frac{P \vdash P' \quad Q \vdash Q'}{\{\mathcal{K} : \exists \bar{y}.\ P \rightsquigarrow Q\} \vdash_{\mathrm{I}} \{\mathcal{K} : \exists \bar{y}.\ P' \rightsquigarrow Q'\}}$$

Action Model Closure Let us now turn to the definition of action model closure, as informally introduced at the beginning of this section. First, we need to revisit the effect of actions to take into account the splitting of the global shared state into a subjective state s and a context r.

Definition 22 (Action application (cont.))**.** The *application* of action $a = (p, q, c)$ on the subjective state s together with the context r, written $a[s, r]$, is defined provided that $a[s \circ r]$ is defined. When that is the case, we write $a[s, r]$ for

$$\left\{ (q \circ s', r') \;\middle|\; \begin{array}{l} p = p_s \circ p_r \wedge p_s > \mathbf{0}_{\mathsf{L}} \\ \wedge\ s = p_s \circ s' \wedge r = p_r \circ r' \end{array} \right\} \\ \cup \{(s, q \circ r') \mid r = p \circ r'\}$$

We observe that this new definition and Def. 8 are linked in the following way:

$$\forall s', r' \in a[s, r].\, s' \circ r' = a[s \circ r]$$

In our informal description of action model closure in Def. 16 we stated that some actions must be *reflected* in some action models. Intuitively, an action is reflected in an action model if for every state in which the action can take place, the action model includes an action with a similar effect that can also occur in that state. In other words, $a = (p, q, c)$ is reflected in $\mathcal{I}$ from a state l if whenever a is enabled in l (i.e. $p \circ c \leq l$), then there exists an action $a' = (p, q, c') \in \mathcal{I}$ (with the same pre- and post states p and q) that is also enabled in l (i.e. $p \circ c' \leq l$). We proceed with the definition of action reflection.

Definition 23. An action $a = (p, q, c)$ is *reflected* in a set of actions A from a state l, written $\mathit{reflected}(a, l, A)$, if

$$\forall r.\, p \circ c \leq l \circ r \Rightarrow \exists c'.\, (p, q, c') \in A \wedge p \circ c' \leq l \circ r$$

Let us now give the formal definition of action model closure. For each condition mentioned in Def. 16, we annotate which part of the definition implements them. Given an action $a = (p, q, c)$, we write

$$\mathit{enabled}(a, g) \stackrel{\text{def}}{=} \mathit{potential}(a, g) \wedge p \circ c \leq g$$

That is, a can actually happen in g since g holds all the resources in both the pre-state of a and its condition.

Definition 24 (Action model closure)**.** An action model $\mathcal{I}$ is *closed* under a subjective state s, context r, and action model $\mathcal{I}'$, written $\mathcal{I}{\downarrow}\,(s, r, \mathcal{I}')$, if $\mathcal{I}{\downarrow_n}\,(s, r, \mathcal{I}')$ for all $n \in \mathbb{N}$, where $\downarrow_n$ is defined recursively as follows:

$$\begin{aligned}
&\mathcal{I}{\downarrow_0}\,(s, r, \mathcal{I}') \stackrel{\text{def}}{\iff} \mathit{true} \\
&\mathcal{I}{\downarrow_{n+1}}\,(s, r, \mathcal{I}') \stackrel{\text{def}}{\iff} \\
&\quad (\forall \kappa.\, \forall a \in \mathcal{I}'(\kappa).\, (\mathit{potential}(a, s \circ r) \Rightarrow \mathit{reflected}(a, s \circ r, \mathcal{I}(\kappa)) \quad (1) \\
&\quad (\forall \kappa.\, \forall a \in \mathcal{I}(\kappa).\, \mathit{potential}(a, s \circ r) \Rightarrow \\
&\qquad (\mathit{reflected}(a, s \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s)) \quad (2) \\
&\qquad \wedge \forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_n}\,(s', r', \mathcal{I}') \quad (3)
\end{aligned}$$

The above states that the $\mathcal{I}$ is closed under $(s, r, \mathcal{I}')$ if the closure relation holds for any number of steps $n \in \mathbb{N}$ where

- s denotes the subjective view of the shared state;

- r denotes the context;
- $s \circ r$ captures the entire shared state;
- a step corresponds to the occurrence of an action as prescribed in $\mathcal{I}$ which may or may not be found in $\mathcal{I}'$.

The relation is satisfied trivially for no steps ($n = 0$). On the other hand, for an arbitrary $n \in \mathbb{N}$ the relation holds iff

1. for any action a in $\mathcal{I}'$, where a is potentially enabled in $s \circ r$, then a is reflected in $\mathcal{I}$; and

2. for any action a in $\mathcal{I}$, where a is potentially enabled in $s \circ r$, then *either*:

   (a) a is reflected in $\mathcal{I}'$ (a is known to the subjective view), and $\mathcal{I}$ is also closed under any subjective state s' and context r' resulting from application of a ($(s', r') \in a[s, r]$); *or*

   (b) a does not affect the subjective state s (a is not visible in s), and $\mathcal{I}$ is also closed under any subjective state s' and context r' resulting from application of a ($(s', r') \in a[s, r]$).

Recall from the semantics of the assertions (Def. 17) that $\mathcal{I}'$ corresponds to the interpretation of an interference assertion I in a subjective view. The first condition thus asserts that the actions in the subjective action model $\mathcal{I}'$ are reflected in $\mathcal{I}$.

Note that in the last item above (bb), since a is not visible in s, given any $(s', r') \in a[s, r]$, from the definition of action application we then know $s' = s$.

We make further observations about this definition. First, property (3) makes our assertions robust with respect to future extensions of the shared state, where potentially enabled actions may be become enabled using additional catalyst that is not immediately present. Second, $\mathcal{I}'$ (and thus interference assertions) need not reflect actions that have no visible effect on the subjective state.

This completes the definition of the semantics of assertions. We can now show that the logical principles of CoLoSL are sound. The proof of the Extend principle will be delayed until the following chapter, where $\Rrightarrow$ is defined.

Lemma 2. The logical implications Copy, Forget, and Merge are *valid*; *i.e.* true of all worlds and logical interpretations.

Proof. The case of Copy is straightforward from the semantics of assertions. In order to show that the Forget implication is valid, it suffices to show:

$$\forall \iota \in \mathsf{LEnv}.\ \left\{w \mid w, \iota \vDash \boxed{P * Q}_I\right\} \subseteq \left\{w \mid w, \iota \vDash \boxed{P}_I\right\}$$

We first demonstrate that, whenever an action model is closed under a subjective state $s_1 \circ s_2$ and context r, then it is also closed under the smaller subjective state s_1, and the larger context extended with the forgotten state. That is,

$$\begin{aligned}\forall s_1, s_2, r \in \mathsf{LState}.\, \forall \mathcal{I}, \mathcal{I}' \in \mathsf{AMod}.& \\ \mathcal{I}{\downarrow}\,(s_1 \circ s_2, r, \mathcal{I}') \implies \mathcal{I}{\downarrow}\,(s_1, s_2 \circ r, \mathcal{I}')&\end{aligned}$$

This is formalised in Lemma 6 of Appendix A. We then proceed to establish the desired result by calculation:

$$\begin{aligned}\left\{w \mid w, \iota \vDash \boxed{P * Q}_I\right\} &= \left\{ \left((\mathbf{0}_\mathsf{L}, (s \circ r), \mathcal{I}\right) \;\middle|\; \begin{array}{l} \exists s_p, s_q 7.\, s = s_p \circ s_q \\ \wedge s_p, \iota \vDash_{(s \circ r), \mathcal{I}} P \\ \wedge s_q, \iota \vDash_{(s \circ r), \mathcal{I}} Q \\ \wedge \mathcal{I}{\downarrow}\,(s_p \circ s_q,\ r, \langle\!|I|\!\rangle_\iota) \end{array} \right\} \\ \text{By Lemma 6} \quad &\subseteq \left\{ \left((\mathbf{0}_\mathsf{L}, (s \circ r), \mathcal{I}\right) \;\middle|\; \begin{array}{l} \exists s_p, s_q.\, s = s_p \circ s_q \\ \wedge s_p, \iota \vDash_{(s \circ r), \mathcal{I}} P \\ \wedge s_q, \iota \vDash_{(s \circ r), \mathcal{I}} Q \\ \wedge \mathcal{I}{\downarrow}\,(s_p,\ s_q \circ r, \langle\!|I|\!\rangle_\iota) \end{array} \right\} \\ &\subseteq \left\{ \left((\mathbf{0}_\mathsf{L}, (s \circ r), \mathcal{I}\right) \;\middle|\; \begin{array}{l} s, \iota \vDash_{(s \circ r), \mathcal{I}} P \\ \wedge \mathcal{I}{\downarrow}\,(s,\ r, \langle\!|I|\!\rangle_\iota) \end{array} \right\} \\ &= \left\{w \mid w, \iota \vDash \boxed{P}_I\right\}\end{aligned}$$

as required.

Note that, the version of Forget where predicates are conjoined using $\uplus$ is also valid for all P, Q, and I, as shown by the following derivation, where the first implication follows from the properties of $\uplus$.

$$\boxed{P \uplus Q}_I \overset{\text{Weaken}}{\Rightarrow} \boxed{P * \mathit{true}}_I \overset{\text{Forget}}{\Rightarrow} \boxed{P}_I$$

Similarly, to show that the Merge implication is valid, it suffices to show:

$$\forall \iota \in \mathsf{LEnv}.\ \left\{w \mid w, \iota \vDash \boxed{P}_{I_1} * \boxed{Q}_{I_2}\right\} \subseteq \left\{w \mid w, \iota \vDash \boxed{P \uplus Q}_{I_1 \cup I_2}\right\}$$

We first demonstrate that, whenever an action model closure relation holds for two (potentially) different subjective states (that may overlap), subject

to two (potentially) different interference relations, then the closure relation also holds for the combined subjective states and their associated interference relations. That is:

$$\begin{array}{l}\forall s_p, s_q, s_c, r \in \mathsf{LState}.\, \forall \mathcal{I}, \mathcal{I}_1, \mathcal{I}_2 \in \mathsf{AMod}. \\ \quad \mathcal{I}{\downarrow}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \;\wedge\; \mathcal{I}{\downarrow}\,(s_q \circ s_c, s_p \circ r, \mathcal{I}_2) \implies \\ \qquad \mathcal{I}{\downarrow}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2)\end{array}$$

This is formalised in Lemma 7 of Appendix A. We then proceed to establish the desired result by calculation:

$$\begin{array}{rl} & \left\{ w \mid w, \iota \vDash \boxed{P}_{I_1} * \boxed{Q}_{I_2} \right\} \\ = & \left\{ (\mathbf{0}_{\mathsf{L}}, s \circ r, \mathcal{I}) \;\middle|\; \begin{array}{l} \exists s_p, s_c, s_q.\, s = s_p \circ s_c \circ s_q \\ \wedge r \in \mathsf{LState} \\ \wedge (s_p \circ s_c), \iota \vDash_{(s \circ r), \mathcal{I}} P \\ \wedge (s_q \circ s_c), \iota \vDash_{(s \circ r), \mathcal{I}} Q \\ \wedge \mathcal{I}{\downarrow}\,(s_p \circ s_c,\ s_q \circ r, \langle\!| I_1 |\!\rangle_\iota) \\ \wedge \mathcal{I}{\downarrow}\,(s_c \circ s_q,\ s_p \circ r, \langle\!| I_2 |\!\rangle_\iota) \end{array} \right\} \\ \text{By Lemma 7} \subseteq & \left\{ (\mathbf{0}_{\mathsf{L}}, s \circ r, \mathcal{I}) \;\middle|\; \begin{array}{l} \exists s_p, s_c, s_q.\, s = s_p \circ s_c \circ s_q \\ \wedge r \in \mathsf{LState} \\ \wedge (s_p \circ s_c), \iota \vDash_{(s \circ r), \mathcal{I}} P \\ \wedge (s_q \circ s_c), \iota \vDash_{(s \circ r), \mathcal{I}} Q \\ \wedge \mathcal{I}{\downarrow}\,(s_p \circ s_c \circ s_q, r, \langle\!| I_1 \cup I_2 |\!\rangle_\iota) \end{array} \right\} \\ \subseteq & \left\{ (\mathbf{0}_{\mathsf{L}}, s \circ r, \mathcal{I}) \;\middle|\; \begin{array}{l} s, \iota \vDash_{(s \circ r), \mathcal{I}} P \mathbin{-\!\!\circledast} Q \\ \wedge r \in \mathsf{LState} \\ \wedge \mathcal{I}{\downarrow}\,(s, r, \langle\!| I_1 \cup I_2 |\!\rangle_\iota) \end{array} \right\} \\ = & \left\{ w \mid w, \iota \vDash \boxed{P \mathbin{-\!\!\circledast} Q}_{I_1 \cup I_2} \right\} \end{array}$$

Finally, to show that the SHIFT implication is valid, it suffices to show that for all $\iota \in \mathsf{LEnv}$,

$$\begin{array}{ll} \text{if} & \langle\!| I |\!\rangle_\iota \sqsubseteq^{\{s \mid s, \iota \vDash_{\mathsf{SL}} P\}} \langle\!| I' |\!\rangle_\iota \\ \text{then} & \left\{ w \mid w, \iota \vDash \boxed{P}_I \right\} \subseteq \left\{ w \mid w, \iota \vDash \boxed{P}_{I'} \right\} \end{array}$$

Pick an arbitrary $w_1 = (l, g, \mathcal{I})$ such that

$$w_1, \iota \vDash \boxed{P}_I \tag{3.1}$$

$$\langle\!| I |\!\rangle_\iota \sqsubseteq^{\{s \mid s, \iota \vDash_{\mathsf{SL}} P\}} \left\langle\!\middle| I' \right\rangle_\iota \tag{3.2}$$

We are then required to show

$$w_1, \iota \vDash \boxed{P}_{I'} \tag{3.3}$$

From (3.1) and definition of w_1 we know that there exist $s, r \in \mathsf{LState}$ such that

$$l = \mathbf{0}_{\mathsf{L}} \wedge g = s \circ r \tag{3.4}$$
$$s, \iota \vDash_{g,\mathfrak{I}} P \tag{3.5}$$
$$\mathfrak{I}\downarrow (s, r, \langle\!| I |\!\rangle_{\iota}) \tag{3.6}$$

From the following lemma and (3.5) we know that

$$s, \iota \vDash_{\mathsf{SL}} P \tag{3.7}$$

Lemma 3. for all $P \in \mathsf{Assn}$, $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathfrak{I} \in \mathsf{AMod}$:

$$s, \iota \vDash_{g,\mathfrak{I}} P \implies s, \iota \vDash_{\mathsf{SL}} P$$

Proof. By induction on the structure of assertion P. Pick an arbitrary $\iota \in \mathsf{LEnv}$, then

Case $P \stackrel{\text{def}}{=} A$ Immediate.
Case $P \stackrel{\text{def}}{=} P_1 \implies P_2$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathfrak{I} \in \mathsf{AMod}$ such that

$$s, \iota \vDash_{g,\mathfrak{I}} P_1 \implies P_2 \tag{3.8}$$
$$\forall \iota, s, g, \mathfrak{I}.\, s, \iota \vDash_{g,\mathfrak{I}} P_1 \implies s, \iota \vDash_{\mathsf{SL}} P_1 \tag{I.H1}$$
$$\forall \iota, s, g, \mathfrak{I}.\, s, \iota \vDash_{g,\mathfrak{I}} P_2 \implies s, \iota \vDash_{\mathsf{SL}} P_2 \tag{I.H2}$$

From (3.8) and the definition of $\vDash_{g,\mathfrak{I}}$ we know $s, \iota \vDash_{g,\mathfrak{I}} P_1$ implies $P_2 \vDash_{g,\mathfrak{I}}$; consequently, from (I.H1) and (I.H2) we have: $s, \iota \vDash_{\mathsf{SL}} P_1$ implies $s, \iota \vDash_{\mathsf{SL}} P_2$. Thus, from the definition of $\vDash_{\mathsf{SL}}$ we have:

$$s, \iota \vDash_{\mathsf{SL}} P_1 \implies P_2$$

as required.

Cases $P \stackrel{\text{def}}{=} \exists x.\, P'$**;** $P \stackrel{\text{def}}{=} P_1 \ast P_2$**;** $P \stackrel{\text{def}}{=} P_1 \mathbin{\text{\Lightning}} P_2$
These cases are analogous to the previous case and are omitted here.

Case $P \stackrel{\text{def}}{=} \boxed{P'}_I$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathfrak{I} \in \mathsf{AMod}$ such that

$$s, \iota \vDash_{g,\mathfrak{I}} \boxed{P'}_I \tag{3.9}$$

From (3.9) and the definition of $\vDash_{g,\mathfrak{I}}$ we know $s, g, \mathfrak{I} \vDash \boxed{P'}_I$ and hence, from the definition of $\vDash$, we have $s = \mathbf{0}_{\mathsf{L}}$. Consequently, from the definition of $\vDash_{\mathsf{SL}}$ we have:

$$s, \iota \vDash_{\mathsf{SL}} \boxed{P'}_I$$

as required. □

Consequently, from the definition of $\sqsubseteq$, (3.2) and (3.7) we have:

$$\langle\!|I|\!\rangle_\iota \sqsubseteq^{\{s\}} \langle\!|I'|\!\rangle_\iota \tag{3.10}$$

and thus from (3.6), (3.10) and Lemma 9 we have:

$$\mathfrak{I}{\downarrow}\left(s, r, \langle\!|I'|\!\rangle_\iota\right) \tag{3.11}$$

Consequently, from 3.4, 3.5, 3.11 and the definition of $\vDash$ we have

$$w, \iota \vDash \boxed{P}_{I'}$$

as required. □

4. CoLoSL Program Logic

This section introduces the core concepts behind our program logic. We start by defining what the rely and guarantee conditions of each thread are in terms of their action models. This allows us to define *stability* against rely conditions, *repartitioning*, which logically represents a thread's atomic actions (and have to be in the guarantee condition), and *semantic implication*. Equipped with these notions, we can justify the SHIFT and EXTEND principles.

4.1 Environment Semantics

Rely The rely relation represents potential interferences from the environment. Although the rely will be different for every program (and indeed, every thread), it is always defined in the same way, which we can break down into three kinds of possible interferences. In the remainder of this section, given a logical state $l = (m, \kappa)$, we write l_{M}, l_{K} for $fst(l)$ and $snd(l)$, respectively.

The first relation, R^{e}, extends the shared state g with new state g', along with a new interference $\mathcal{I}'$ on g'. We proceed with the definition of *action model extension* that captures the extension of the action model in such a way that respects all previous subjective views of a world; that is, the action model closure relation is preserved.

Definition 25 (Action model extension)**.** An action model $\mathcal{I}'$ *extends* $(g, \mathcal{I})$ with (g'), written $\mathcal{I}' \uparrow^{(g')} (g, \mathcal{I})$, iff for all $\mathcal{I}_0 \subseteq \mathcal{I}$ and s, r such that $s \circ r = g$:

$$\mathcal{I}{\downarrow}\,(s, r, \mathcal{I}_0) \implies \mathcal{I} \cup \mathcal{I}'{\downarrow}\,\big(s, r \circ g', \mathcal{I}_0\big)$$

Then the extension rely, R^{e}, is defined as

$$R^{\mathsf{e}} \stackrel{\text{def}}{=} \left\{ \begin{pmatrix} (l, g, \mathcal{I}), \\ (l, g \circ g', \mathcal{I} \cup \mathcal{I}') \end{pmatrix} \;\middle|\; \begin{array}{l} g'_{\mathsf{K}} \prec \mathsf{dom}\,(\mathcal{I}') \cup \mathsf{dom}\,(\mathcal{I}) \wedge \\ g' \copyright \mathcal{I}' \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \end{array} \right\}$$

The second and last kind of interference is the *update* of the global state according to actions in the global action model whose capability is "owned by the environment", *i.e.*, nowhere to be found in the current local and shared states.

$$R^{\mathsf{u}} \stackrel{\text{def}}{=} \left\{ \left((l, g, \mathcal{I}),\ (l, g', \mathcal{I}) \right) \middle| \exists \kappa.\ (l_{\mathsf{K}} \bullet_{\mathbb{K}} g_{\mathsf{K}}) \sharp \kappa \wedge (g, g') \in \lceil \mathcal{I} \rceil (\kappa) \right\}$$

where

$$\lceil \mathcal{I} \rceil (\kappa) \stackrel{\text{def}}{=} \{ (p \circ r, q \circ r) \mid (p, q) \in \mathcal{I}(\kappa) \wedge r \in \mathsf{LState} \}$$

Definition 26 (Rely)**.** The *rely* relation $R : \mathcal{P}(\mathsf{World} \times \mathsf{World})$ is defined as follows, where $(\cdot)^*$ denotes the reflexive transitive closure:

$$R \stackrel{\text{def}}{=} \left(R^{\mathsf{u}} \cup R^{\mathsf{e}} \right)^*$$

The rely relation enables us to define the stability of assertions with respect to the environment actions.

Definition 27 (Stability)**.** An assertion P is *stable* ($\mathsf{Stable}(P)$) if, for all $\iota \in \mathsf{LEnv}$ and $w, w' \in \mathsf{World}$, if $w, \iota \vDash P$ and $(w, w') \in R$, then $w', \iota \vDash P$.

Proving that an assertion is stable is not always obvious, in particular when there are numerous transitions to consider (all those in R^{e}, R^{u}, R^{s}); as it turns out, we only need to check stability against update actions in R^{u}, as expressed by the following lemma.

Lemma 4 (Stability)**.** If an assertion P is stable with respect to actions in R^{u}, then it is stable. That is, for all $w, w' \in \mathsf{World}$, $\iota \in \mathsf{LEnv}$, and $P \in \mathsf{Assn}$,

$$\text{if } w, \iota \vDash P \wedge (w, w') \in R \wedge \left(\forall w_1, w_2.\ \begin{pmatrix} w_1, \iota \vDash P \wedge \\ (w_1, w_2) \in R^{\mathsf{u}} \end{pmatrix} \Longrightarrow w_2, \iota \vDash P \right)$$

then $w', \iota \vDash P$

Proof. Given any arbitrary $n \in \mathbb{N}$, let $S^n \in \mathsf{World} \times \mathsf{World}$ denote a binary relation on worlds defined inductively as follows.

$$S^0 \stackrel{\text{def}}{=} R^{\mathsf{e}} \cup R^{\mathsf{u}} \cup \{ (w, w) \mid w \in \mathsf{World} \}$$
$$S^{n+1} \stackrel{\text{def}}{=} \left\{ (w, w') \mid (w, w'') \in S^0 \wedge (w'', w) \in S^n \right\}$$

Let $S \in \mathsf{World} \times \mathsf{World}$ denote a binary relation on worlds defined as follows.

$$S \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} S^i$$

From the definition of the rely relation on worlds (R) we then have $R = S$. It thus suffices to show that for all $n \in \mathbb{N}$, $w, w' \in \mathsf{World}$, $\iota \in \mathsf{LEnv}$ and $P \in \mathsf{Assn}$,

$$\text{if } w, \iota \vDash P \wedge (w, w') \in S^n \wedge \left(\forall w_1, w_2.\ \begin{pmatrix} w_1, \iota \vDash P \wedge \\ (w_1, w_2) \in R^{\mathsf{u}} \end{pmatrix} \Longrightarrow w_2, \iota \vDash P\right)$$
$$\text{then } w', \iota \vDash P$$

We proceed by induction on n.

Base case $n = 0$
Pick an arbitrary $w, w' \in \mathsf{World}$, $\iota \in \mathsf{LEnv}$ and $P \in \mathsf{Assn}$ such that

$$(w, w') \in S^0 \tag{4.1}$$
$$w, \iota \vDash P \wedge \left(\forall w_1, w_2.\ \left(w_1, \iota \vDash P \wedge (w_1, w_2) \in R^{\mathsf{u}}\right) \Longrightarrow w_2, \iota \vDash P\right) \tag{4.2}$$

We are then required to show $w', \iota \vDash P$. From (4.1) there are then three cases to consider:

1. If $(w, w') \in \{(w, w) \mid w \in \mathsf{World}\}$, then $w' = w$ and from (4.1) we trivially have $w', \iota \vDash P$ as required.
2. If $(w, w') \in R^{\mathsf{e}}$ then from (4.2) and Lemma 17 we have $w', \iota \vDash P$ as required.
3. If $(w, w') \in R^{\mathsf{u}}$ then from (4.2) we trivially have $w', \iota \vDash P$ as required.

Inductive case
Pick an arbitrary $w, w' \in \mathsf{World}$, $\iota \in \mathsf{LEnv}$ and $P \in \mathsf{Assn}$ such that

$$(w, w') \in S^{n+1} \tag{4.3}$$
$$w, \iota \vDash P \wedge \left(\forall w_1, w_2.\ \begin{pmatrix} w_1, \iota \vDash P \wedge \\ (w_1, w_2) \in R^{\mathsf{u}} \end{pmatrix} \Longrightarrow w_2, \iota \vDash P\right) \tag{4.4}$$

We are then required to show $w', \iota \vDash P$ provided that

$$\forall m \leq n.\, \forall w_3, w_4 \in \mathsf{World}.$$
$$\text{if } w_3, \iota \vDash P \wedge (w_3, w_4) \in S^m \wedge \left(\forall w_1, w_2.\ \begin{pmatrix} w_1, \iota \vDash P \wedge \\ (w_1, w_2) \in R^{\mathsf{u}} \end{pmatrix} \Longrightarrow w_2, \iota \vDash P\right)$$
$$\text{then } w_4, \iota \vDash P \tag{I.H.}$$

From (4.3) and the definition of S^{n+1} we know there exists $w'' \in \mathsf{World}$ such that

$$(w, w'') \in S^0 \tag{4.5}$$
$$(w'', w') \in S^n \tag{4.6}$$

From (4.5), (4.4) and the base case we know

$$w'', \iota \vDash P \tag{4.7}$$

Consequently, from (4.4), (4.6), (4.7) and (I.H.) we have

$$w', \iota \vDash P$$

as required. □

We provide a number of syntactic judgements in Fig. 4.1 that allows us to ascertain the stability of an assertion without performing the necessary semantic checks. These judgements reduce stability checks to logical entailments. We appeal to the following auxiliary definition in our stability judgements.

Definition 28 (Combination)**.** The *combination* of an assertion $P \stackrel{\text{def}}{=} \exists \bar{x}.\, P'$ describing the current state, and an assertion $Q \stackrel{\text{def}}{=} \exists \bar{y}.\, Q'$ describing an action precondition, $\mathsf{c}\,(.,.) : \mathsf{Assn} \times \mathsf{Assn} \to \mathsf{LAssn}$, is defined as follows. We assume that P and Q are in the prenex normal form with no bound variables in P' and Q'.

$$\begin{aligned} \mathsf{c}\,(P, Q) \stackrel{\text{def}}{=} \quad & \exists \bar{x}, \bar{y}.\, p * (p' \circledast q \circledast q') \\ & \text{where } (p, p') = \mathsf{ub}\,(\mathsf{f}\,(P')) \text{ and } (q, q') = \mathsf{ub}\,(\mathsf{f}\,(Q')) \end{aligned}$$

Guarantee We now define the guarantee relation that describes all possible updates the current thread can perform. In some sense, the guarantee relation is the dual of rely: the actions in the guarantee of one thread are included in the rely of concurrently running threads. Thus, it should come as no surprise that transitions in the guarantee can be categorised using three categories which resonate with those of the rely. The *extension* guarantee is similar to the extension rely except that new capabilities corresponding to the new

$$\frac{}{\mathsf{Stable}\,(p)} \qquad \frac{\mathsf{Stable}\,(P) \quad \mathsf{Stable}\,(Q)}{\mathsf{Stable}\,(P \odot Q)} \qquad \frac{\mathsf{Stable}\,(P)}{\mathsf{Stable}\,(\exists x.\, P)}$$

$$\frac{\mathsf{St}\,(P, Q) \quad \mathsf{St}\,(Q, P)}{\mathsf{Stable}\,(P * Q)}$$

$$\frac{\mathsf{Stable}\,(P)}{\mathsf{St}\,(P, R)} \qquad \frac{\mathsf{St}\,(P, Q * R) \quad \mathsf{St}\,(Q, P * R)}{\mathsf{St}\,(P * Q, R)} \qquad \frac{\mathsf{St}\,(P, R)}{\mathsf{St}\,(P, R * R')}$$

$$\frac{\mathsf{St}\,(P, R) \quad \mathsf{St}\,(Q, R)}{\mathsf{St}\,(P \odot Q, R)} \qquad \frac{\mathsf{St}\,(P, R)}{\mathsf{St}\,(\exists x.\, P, R)} \qquad \frac{\mathsf{St}\left(\boxed{p}_{I'}, R\right) \quad I' \sqsubseteq^{p} I}{\mathsf{St}\left(\boxed{p}_{I}, R\right)}$$

$$\frac{\mathsf{S}\left(p, I, R * \boxed{p}_{I}\right)}{\mathsf{St}\left(\boxed{p}_{I}, R\right)}$$

$$\frac{\mathsf{S}\,(p, I, R)}{\mathsf{S}\,(p, I, R * Q)} \qquad \frac{\mathsf{S}\,(p, I_1, R) \quad \mathsf{S}\,(p, I_2, R)}{\mathsf{S}\,(p, I_1 \cup I_2, R)} \qquad \frac{I \vdash_{\mathrm{I}} I' \quad \mathsf{S}\,(p, I', R)}{\mathsf{S}\,(p, I, R)}$$

$$\frac{\mathcal{K} * \bigotimes R \vdash_{\mathsf{SL}} \mathit{false}}{\mathsf{S}\,(p, \{\mathcal{K} : \exists \bar{y}.\, q_1 \rightsquigarrow q_2\}, R)} \qquad \frac{\mathsf{c}\,(R, q_1) \vdash_{\mathsf{SL}} \mathit{false}}{\mathsf{S}\,(p, \{\mathcal{K} : q_1 \rightsquigarrow q_2\}, R)}$$

$$\frac{(q_1 \mathrel{-\!\!\circledast} \mathsf{c}\,(R, q_1)) * q_2 \vdash_{\mathsf{SL}} p * \mathit{true}}{\mathsf{S}\,(p, \{\mathcal{K} : q_1 \rightsquigarrow q_2\}, R)}\ !!!$$

Figure 4.1: Stability judgements where $P, Q, Q_1, Q_2, R \in \mathsf{FAssn}$; $p, q_1, q_2 \in \mathsf{LAssn}$ and $\dot{\in}\{\vee, *, \circledast\}$.

shared resources are materialised in the local and shared state, and a part of the local state is moved into the shared state:

$$G^{\mathsf{e}} \stackrel{\text{def}}{=} \left\{ \begin{pmatrix} (l \circ l', g, \mathcal{I}), \\ l \circ (\mathbf{0}_{\mathbb{M}}, \kappa_1), g \circ g', \mathcal{I} \cup \mathcal{I}' \end{pmatrix} \;\middle|\; \begin{array}{l} g' = l' \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2) \wedge \\ \kappa_1 \bullet_{\mathbb{K}} \kappa_2 \prec \mathsf{dom}\,(\mathcal{I}') \wedge \\ \kappa_1 \bullet_{\mathbb{K}} \kappa_2 \perp \mathsf{dom}\,(\mathcal{I}) \wedge \\ g' \mathbin{©} \mathcal{I}' \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \end{array} \right\}$$

The current thread may at any point extend the shared state with some of its locally held resources l'; introduce new interference to describe how the new resources may be mutated ($\mathcal{I}'$) and generate new capabilities ($\kappa_1 \bullet_{\mathbb{K}} \kappa_2 \prec \mathsf{dom}\,(\mathcal{I}')$) that facilitate the new interference, with the proviso that the new capabilities are fresh ($\kappa_1 \bullet_{\mathbb{K}} \kappa_2 \perp \mathsf{dom}\,(\mathcal{I})$). The last two conjuncts enforce closure of the new action models and can be justified as in the case of R^{e}.

The *update guarantee* G^{u} is more involved than its R^{u} counterpart, because updates in the guarantee may move resources from the local state into the shared state (similarly to extensions above) at the same time that it mutates them as prescribed by an enabled action. Intuitively, we want to enforce that resources are not created "out of thin air" in the process. This can be expressed as preserving the *orthogonal* of the combination of the local and global states, *i.e.*, the set of states compatible with that combination.

Definition 29 (Orthogonal)**.** Given any separation algebra $(\mathbb{B}, \bullet_{\mathbb{B}}, \mathbf{0}_{\mathbb{B}})$, and an element $b \in \mathbb{B}$, its *orthogonal* $(.)^{\perp}_{\mathbb{B}} : \mathbb{B} \to \mathcal{P}\,(\mathbb{B})$, is defined as the set of all elements in $\mathbb{B}$ that are compatible with it:

$$(b)^{\perp}_{\mathbb{B}} \stackrel{\text{def}}{=} \left\{ b' \mid b \sharp b' \right\}$$

The *update guarantee* G^{u} can then be defined as follows. When updating the shared state, thread are not allowed to introduce new capabilities, as this can only be achieved when extending the shared state (through G^{e}).

$$G^{\mathsf{u}} \stackrel{\text{def}}{=} \left\{ ((l, g, \mathcal{I}), (l', g', \mathcal{I})) \;\middle|\; \begin{array}{l} ((l' \circ s')_{\mathsf{K}})^{\perp}_{\mathbb{K}} = ((l \circ g)_{\mathsf{K}})^{\perp}_{\mathbb{K}} \wedge \\ \left(\begin{array}{l} g = g' \vee \\ \left(\begin{array}{l} \exists \kappa \leq l_{\mathsf{K}}.\, (g, g') \in \lceil \mathcal{I} \rceil\,(\kappa) \wedge \\ ((l \circ g)_{\mathsf{M}})^{\perp}_{\mathbb{M}} = ((l' \circ g')_{\mathsf{M}})^{\perp}_{\mathbb{M}} \end{array} \right) \end{array} \right) \end{array} \right\}$$

Definition 30 (Guarantee)**.** The *guarantee* relation $G : \mathcal{P}(\mathsf{World} \times \mathsf{World})$ is defined as

$$G \stackrel{\text{def}}{=} \left(G^{\mathsf{u}} \cup G^{\mathsf{e}} \right)^*$$

Using the guarantee relation, we introduce the notion of *repartitioning* $P \Rrightarrow^{\{R_1\}\{R_2\}} Q$. This relation holds whenever, from any world satisfying P, if whenever parts of the composition of its local and shared states that satisfies R_1 is exchanged for one satisfying R_2, it is possible to split the resulting logical state into a local and shared part again, in such a way that the resulting transition is in G.

Definition 31 (Repartitioning)**.** We write $P \Rrightarrow^{\{R_1\}\{R_2\}} Q$ if, for every $\iota \in \mathsf{LEnv}$, and world w_1 such that $w_1, \iota \vDash P$, there exists states $m_1, m' \in \mathbb{M}$ such that $(m_1, \emptyset), \iota \vDash_{\mathsf{SL}} R_1$ and

- $m_1 \bullet_{\mathbb{M}} m' = (\downarrow(w_1))_{\mathsf{M}}$; and
- for every m_2 where $(m_2, \emptyset), \iota \vDash_{\mathsf{SL}} R_2$, there exists a world w_2 such that $w_2, \iota \vDash Q$; and
  - $m_2 \bullet_{\mathbb{M}} m' = (\downarrow(w_2))_{\mathsf{M}}$; and
  - $(w_1, w_2) \in G$

We write $P \Rrightarrow Q$ for $P \Rrightarrow^{\{\mathsf{emp}\}\{\mathsf{emp}\}} Q$, in which case the repartitioning has no "side effect" and simply shuffles resources around between the local and shared state or modifies the action models. This is the case for (Shift) and (Extend), whose proof will be given in the next section.

4.1.1 Interference Manipulations

In this section we formalise the requirements of the Extend and Shift semantic implications and show that they are valid.

Shared State Extension When extending the shared state using currently owned local resources, one specifies a new interference assertion over these newly shared resources. While in CoLoSL the new interferences may mention parts of the shared state beyond the newly added resources (in particular the existing shared state), they must not allow visible updates to those parts, so as not to invalidate other threads' views of existing resources. We thus impose a locality condition on the newly added behaviour to ensure sound extension of the shared state, similarly to the confinement constraint of local fences of Def. 7. We first motivate this constraint with an example.

Example 1. Let $P \stackrel{\text{def}}{=} x \mapsto 1 * \boxed{y \mapsto 1 \vee y \mapsto 2}_I$ denote the view of the current thread with $I \stackrel{\text{def}}{=} (\mathsf{b} : \{y \mapsto 1 \rightsquigarrow y \mapsto 2\})$. Since the current thread owns the location addressed by x, it can extend the shared state as $Q \stackrel{\text{def}}{=} [\mathsf{a}] * \boxed{(y \mapsto 1 \vee y \mapsto 2) * x \mapsto 1}_{I \cup I'}$ where $I' \stackrel{\text{def}}{=} (\mathsf{a} : x \mapsto 1 \rightsquigarrow x \mapsto 2)$. In extending the shared state, the current thread also extended the interference allowed on the shared state by adding a new action associated with the newly generated capability resource $[\mathsf{a}]$, as given in I', which updates the value of location x. Since location x was previously owned privately by the current thread and was hence not visible to other threads, this new action will not invalidate their view of the shared state, hence this extension is a valid one.

If, on the other hand, I' is replaced with $I'' \stackrel{\text{def}}{=} (\mathsf{a} : \{y \mapsto 1 \rightsquigarrow y \mapsto 3\})$, where location y can be mutated, the situation above is not allowed. Indeed, other threads may rely on the fact that the only updates allowed on location y are done through the $[\mathsf{b}]$ capability as specified in I, and would be spooked by this new possible behaviour they were not aware of (as it is not in I).

In order to ensure sound extension of the shared state, we require in Extend that the newly introduced interferences are confined to the locally owned resources.

Definition 32. A set of states $\mathcal{P}$ *confines* an action model $\mathcal{I}$, written $\mathcal{P} \copyright \mathcal{I}$, if

$$\exists \mathcal{F}.\, \mathcal{P} \subseteq \mathcal{F} \wedge \mathcal{F} \blacktriangleright \mathcal{I}$$

The $P \copyright I$ notation used in Extend is then a straightforward lift of this definition to assertions $P \in \mathsf{Assn}$ and interference assertions $I \in \mathsf{IAssn}$:

$$P \copyright I \iff \forall \iota.\, \{l \mid l, \iota \vDash_{\mathsf{SL}} P\} \copyright \langle\!| I |\!\rangle_\iota$$

Rather than checking the confinement conditions semantically, we present a number of judgements that reduce interference confinement and local fences to logical entailments in Fig. 4.2.

Note that although local fencing judgements are given for a local assertion $f \in \mathsf{LAssn}$, since $\lfloor P \in \mathsf{LAssn}$ and $P \vdash \lfloor P$ given any assertion P (Lemma 1), in the top left confinement judgement one can simply substitute $\lfloor P$ for P. *Mutatis mutandis* for P' and Q' in the left-most judgements of the second row.

Pleasantly, since our local assertions do not contain subjective (boxed) assertions, entailments of the form $f \vdash_{\mathsf{SL}} f'$ are the familiar entailments of standard separation logic.

Definition 33 (Freshness)**.** The capability assertion $\mathcal{K}$ with logical variables $\bar{x}$ is *fresh*, written *fresh* $(\bar{x}, \mathcal{K})$, iff for all $\iota \in \mathsf{LEnv}$ and $K \in \mathcal{P}(\mathbb{K})$ where $\diamond K$, there exists $\kappa \in \mathbb{K}$ such that

$$(\mathbf{0}_{\mathbb{M}}, \kappa), \iota \vDash_{\mathsf{SL}} \exists \bar{x}.\, \mathcal{K} \wedge \kappa \perp K$$

where

$$\kappa \perp K \iff \forall \kappa' \in K.\, \kappa \perp \kappa'$$

Lemma 5. The semantic implication (Extend) is valid.

$$\frac{P \vdash P' \quad P' \blacktriangleright I}{P \mathbin{©} I} \qquad \frac{}{f \blacktriangleright \emptyset} \qquad \frac{f \blacktriangleright I_1 \quad f \blacktriangleright I_2}{f \blacktriangleright I_1 \cup I_2} \qquad \frac{I \vdash_{\mathrm{I}} I' \quad f \blacktriangleright I'}{f \blacktriangleright I}$$

$$\frac{\mathsf{exact}\,(r) \quad f \blacktriangleright \{\mathcal{K}{:}\, p \rightsquigarrow q\}}{f \blacktriangleright \{\mathcal{K} : p * r \rightsquigarrow q * r\}} \qquad \frac{f \circledast p \vdash_{\mathsf{SL}} \mathit{false}}{f \blacktriangleright \{\mathcal{K}{:}\, p \rightsquigarrow q\}}$$

$$\frac{(p \mathrel{-\!\!\circledast} f) * q \vdash_{\mathsf{SL}} f \quad f \Leftrightarrow \bigvee_{i \in I} f_i \quad (\mathsf{precise}\,(f_i) \wedge f_i \circledast p \vdash_{\mathsf{SL}} f_i \ \text{ for } i \in I)}{f \blacktriangleright \{\mathcal{K} : p \rightsquigarrow q\}}$$

Figure 4.2: Confinement/local fencing judgements with $P, Q \in \mathsf{Assn}$; $p, q, r, f \in \mathsf{LAssn}$.

Proof. It suffices to show that for all $\iota \in \mathsf{LEnv}$ and $w_1 \in \mathsf{World}$,

if	$w_1, \iota \vDash P \wedge P * \mathcal{K}_2 \mathbin{©} I \wedge \mathit{fv}(P) \cap \bar{x} = \emptyset \wedge \mathit{fresh}\,(\bar{x}, \mathcal{K}_1 * \mathcal{K}_2)$
then	$\exists w_2.\, w_2, \iota \vDash \exists \bar{x}.\, \mathcal{K}_1 * \boxed{P * \mathcal{K}_2}_I \wedge (\downarrow(w_1))_{\mathsf{M}} = (\downarrow(w_2))_{\mathsf{M}} \wedge (w_1, w_2) \in G$

Pick an arbitrary $\iota \in \mathsf{LEnv}$ and $w_1 = (l, g, \mathcal{I})$ such that

$$w_1, \iota \vDash P \tag{4.8}$$
$$P * \mathcal{K}_2 \mathbin{©} I \tag{4.9}$$
$$\mathit{fresh}\,(\bar{x}, \mathcal{K}_1 * \mathcal{K}_2) \tag{4.10}$$

From (4.10) and the definition of *fresh* we know that

$$\exists \kappa.\, (\mathbf{0}_{\mathbb{M}}, \kappa), \iota \vDash_{\mathsf{SL}} \exists \bar{x}.\, \mathcal{K}_1 * \mathcal{K}_2 \ \wedge \ \kappa \perp \mathsf{dom}\,(\mathcal{I})$$

and consequently there exists $\bar{v} \in \mathsf{Val}$ and $\kappa_1, \kappa_2 \in \mathbb{K}$ such that

$$\kappa_1 \in (\!|\mathcal{K}_1[\bar{v}/\bar{x}]|\!)^K_\iota \wedge \kappa_2 \in (\!|\mathcal{K}_2[\bar{v}/\bar{x}]|\!)^K_\iota \wedge \kappa_1 \bullet_{\mathbb{K}} \kappa_2 \perp \mathsf{dom}\,(\mathcal{I}) \tag{4.11}$$

From (4.9) we have

$$P * \mathcal{K}_2[\bar{v}/\bar{x}] \mathbin{©} I[\bar{v}/\bar{x}] \tag{4.12}$$

From (4.8) and the definition of $\vDash_{\mathsf{SL}}$ we have $l, \iota \vDash_{\mathsf{SL}} P$. Similarly, from (4.11) and the definitions of $\vDash$ and $\vDash_{\mathsf{SL}}$ we have $(\mathbf{0}_{\mathbb{M}}, \kappa_2) \vDash_{\mathsf{SL}} \mathcal{K}_2$. Consequently we have:

$$l \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2), \iota \vDash_{\mathsf{SL}} P * \mathcal{K}_2 \tag{4.13}$$

From the definitions of w_1 and the well-formedness of worlds we know there exists $\mathcal{F} \in \mathcal{P}(\mathsf{LState})$ such that

$$g \in \mathcal{F} \wedge \mathcal{F} \blacktriangleright \mathcal{I} \tag{4.14}$$

Pick $\mathcal{I}_0 \in \mathsf{AMod}$ such that

$$\mathsf{dom}(\mathcal{I}_0) = \{\kappa \mid (\kappa)^{\perp}_{\mathbb{K}} = (\kappa_1 \bullet_{\mathbb{K}} \kappa_2)^{\perp}_{\mathbb{K}}\} \wedge \forall \kappa \in \mathsf{dom}(\mathcal{I}_0) \,.\, \mathcal{I}_0(\kappa) = \emptyset$$

Let $\mathcal{I}'' \stackrel{\text{def}}{=} \langle\!| I[\bar{v}/\bar{x}] |\!\rangle_{\iota}$ and $\mathcal{I}' = \mathcal{I}'' \cup \mathcal{I}_0$; from (4.12), (4.13) and the definition of ⓒ we know there exists $\mathcal{F}' \in \mathcal{P}(\mathsf{LState})$ such that $l \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2) \in \mathcal{F}' \wedge \mathcal{F}' \blacktriangleright \mathcal{I}''$. Consequently from the definitions of $\blacktriangleright$, $\mathcal{I}'$ and $\mathcal{I}_0$ and since $\forall \kappa \in \mathsf{dom}(\mathcal{I}_0) \,.\, \mathcal{I}_0(\kappa) = \emptyset$ we have

$$l \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2) \in \mathcal{F}' \wedge \mathcal{F}' \blacktriangleright \mathcal{I}' \tag{4.15}$$

Let $g' = l \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2)$ and $w_2 = ((\mathbf{0}_{\mathbb{M}}, \kappa_1), g \circ g', \mathcal{I} \cup \mathcal{I}')$. Given the definitions of G and G^{e}, it then suffices to show

$$(\downarrow(w_1))_{\mathsf{M}} = (\downarrow(w_2))_{\mathsf{M}} \tag{4.16}$$

$$\kappa_1 \bullet_{\mathbb{K}} \kappa_2 \prec \mathsf{dom}(\mathcal{I}') \tag{4.17}$$

$$\kappa_1 \bullet_{\mathbb{K}} \kappa_2 \perp \mathsf{dom}(\mathcal{I}) \tag{4.18}$$

$$g' \text{ ⓒ } \mathcal{I}' \tag{4.19}$$

$$\forall s', \mathcal{I}_0 . \mathcal{I}{\downarrow}(s', g - s', \mathcal{I}_0) \implies \mathcal{I} \cup \mathcal{I}'{\downarrow}(s', (g - s') \circ g', \mathcal{I}_0) \tag{4.20}$$

$$w_2, \iota \vDash \exists \bar{x} . \mathcal{K}_1 * \boxed{P * \mathcal{K}_2}_I \tag{4.21}$$

RTS. (4.16) This follows immediately from the definition of $\downarrow(.)$ and the definitions of w_1 and w_2.

RTS. (4.17) This follows trivially from the definitions of $\prec$ and $\mathcal{I}'$ since $\mathcal{I}_0 \subseteq \mathcal{I}'$ and $\kappa_1 \bullet_{\mathbb{K}} \kappa_2 \in \mathsf{dom}(\mathcal{I}_0)$.

RTS. (4.18) This follows trivially from (4.11).

RTS. (4.19) This follows immediately from the definition of g', (4.15) and the definition of ⓒ.

RTS. (4.20) From (4.14), (4.15), the definitions of $\mathcal{I}'$, g' and Lemma 11, we can despatch this obligation.

RTS. (4.21)
From (4.11) and the definition of $\vDash_{\mathsf{SL}}$ we have $(\mathbf{0}_{\mathbb{M}}, \kappa_1), \iota \vDash_{\mathsf{SL}} \mathcal{K}_1[\bar{v}/\bar{x}]$ and consequently by the definition of $\vDash$

$$((\mathbf{0}_{\mathbb{M}}, \kappa_1), g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash \mathcal{K}_1[\bar{v}/\bar{x}] \tag{4.22}$$

Similarly, from (4.11) and the definitions of $\vDash_{\mathsf{SL}}$ and $\vDash$ we have $((\mathbf{0}_{\mathbb{M}}, \kappa_2), g, \mathcal{I}), \iota \vDash \mathcal{K}_2[\bar{v}/\bar{x}]$; consequently from (4.8), the definition of g' and since $fv(P) \cap \bar{x} = \emptyset$, we can conclude

$$(g', g, \mathcal{I}), \iota \vDash (P * \mathcal{K}_2)[\bar{v}/\bar{x}] \tag{4.23}$$

Thus from (4.23), (4.20) - established above - and Lemma 19 we have:

$$g', \iota \vDash_{g \circ g', \mathcal{I} \cup \mathcal{I}'} (P * \mathcal{K}_2)[\bar{v}/\bar{x}] \tag{4.24}$$

On the other hand, from (4.14), (4.15), the definitions of $\mathcal{I}'$, g', Lemma 10 and since $\mathcal{I}' = \mathcal{I}'' \cup \mathcal{I}_0 = \left\langle\!\left| I[\bar{v}/\bar{x}] \right|\!\right\rangle_\iota \cup \mathcal{I}_0$, we have:

$$\mathcal{I} \cup \mathcal{I}' \downarrow \left(g', g, \left\langle\!\left| I[\bar{v}/\bar{x}] \right|\!\right\rangle_\iota\right) \tag{4.25}$$

Consequently from (4.24), (4.25) and the definition of $\vDash$

$$(\mathbf{0}_{\mathsf{L}}, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash (\boxed{P * \mathcal{K}_2}_I)[\bar{v}/\bar{x}] \tag{4.26}$$

From (4.22), (4.26) and the definitions of w_2 and $\vDash$ we have:

$$w_2, \iota \vDash (\mathcal{K}_1 * \boxed{P * \mathcal{K}_2}_I)[\bar{v}/\bar{x}]$$

and consequently from the definition of $\vDash$

$$w_2, \iota \vDash \exists \bar{x}. \mathcal{K}_1 * \boxed{P * \mathcal{K}_2}_I$$

as required. □

Action Shifting Notice that, according to our rely and guarantee conditions, the action model may only grow with time. However, that is not to say that the same holds of interference relations in shared state assertions: as seen in §2, they may forget about actions that are either redundant or not relevant to the current subjective view via *shifting*. As for the action model closure relation (Def. 24), this needs to be the case not only from the current subjective view but also for any evolution of it according to potential actions, both from the thread and the environment. To capture this set of possible

futures, we refine our notion of local fences (Def. 10), which was defined in the context of the global shared state, to the case where we consider only a subjective state within the global shared state. For this, we need to also refine a second time our notion of action application to ignore the context of a subjective state, which as far as a subjective view is concerned could be anything.

Definition 34 (Subjective action application)**.** The *subjective application* of an action a on a logical state s, written $a(s)$ is defined provided that there exists a context r for which $a[s \circ r]$ is defined. When that is the case, we write $a(s)$ for

$$\left\{ s' \;\middle|\; \exists r.\ s \sharp r \wedge (s', -) \in a[s, r] \right\}$$

Note that, in contrast with $a[l]$, only *parts* of the active precondition has to intersect with the subjective view s for $a(s)$ to apply. Thus, we fabricate a context r that is compatible with the subjective view and satisfies the rest of the precondition.

Definition 35 (Fenced action model)**.** An action model $\mathcal{I} \in \mathsf{AMod}$ is *fenced* by $\mathcal{F} \in \mathcal{P}(\mathsf{LState})$, written $\mathcal{F} \rhd \mathcal{I}$, if, for all $l \in \mathcal{F}$ and all $a \in rg(\mathcal{I})$,

$$a(l) \text{ is defined} \Rightarrow a(l) \subseteq \mathcal{F}$$

In contrast with local fences, fences do not require that actions be confined inside the subjective state. We lift the notion of fences to assertions as follows, given $f \in \mathsf{Assn}$ and $I \in \mathsf{IAssn}$:

$$f \rhd I \iff^{\text{def}} \forall \iota.\, \{l \mid l, \iota \vDash_{\mathsf{SL}} F\} \rhd \langle\!| I |\!\rangle_\iota$$

For instance, a possible fence for the interference assertion I_{y} of Fig. 2.2 is denoted by the following assertion.

$$F_{\mathrm{y}} \stackrel{\text{def}}{=} \bigvee_{v=0}^{10} (x \mapsto v * y \mapsto v) \vee (x \mapsto v+1 * y \mapsto v)$$

Definition 36 (Action shifting)**.** Given $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ and $\mathcal{P} \in \mathcal{P}(\mathsf{LState})$, $\mathcal{I}'$ is a *shifting* of $\mathcal{I}$ with respect to $\mathcal{P}$, written $\mathcal{I} \sqsubseteq^{\mathcal{P}} \mathcal{I}'$, if there exists a fence $\mathcal{F}$ such that

$$\begin{array}{l} \mathcal{P} \subseteq \mathcal{F} \wedge \mathcal{F} \rhd \mathcal{I} \wedge \forall l \in \mathcal{F}.\, \forall \kappa. \\ \ (\forall a \in \mathcal{I}'(\kappa).\, \mathit{reflected}(a, l, \mathcal{I}(\kappa))) \wedge \\ \ \forall a \in \mathcal{I}(\kappa).\, a(l) \text{ is defined} \wedge \mathit{visible}(a, l) \Rightarrow \mathit{reflected}(a, l, \mathcal{I}'(\kappa)) \end{array}$$

$$\frac{}{true \rhd I} \qquad \frac{f \blacktriangleright I}{f \rhd I} \qquad \frac{f \rhd I_1 \quad f \rhd I_2}{f \rhd I_1 \cup I_2} \qquad \frac{I \vdash_{\mathrm{I}} I' \quad f \rhd I'}{f \rhd I}$$

$$\frac{\mathsf{exact}\,(r) \quad f \rhd \{\mathcal{K}{:}\, p \rightsquigarrow q\}}{f \rhd \{\mathcal{K} : p * r \rightsquigarrow q * r\}} \qquad \frac{f \rhd I' \quad I' \sqsubseteq^{f} I}{f \rhd I} \qquad \frac{f \cap p \vdash_{\mathsf{SL}} \mathsf{emp}}{f \rhd \{\mathcal{K}{:}\, p \rightsquigarrow q\}}$$

$$\frac{p \cap q \vdash_{\mathsf{SL}} \mathsf{emp} \quad (p \mathrel{-\!\!\circledast} (f \uplus p)) * q \vdash_{\mathsf{SL}} f}{f \rhd \{\mathcal{K} : p \rightsquigarrow q\}}$$

Figure 4.3: Fencing judgements.

The first conjunct expresses the fact that the new action model $\mathcal{I}'$ cannot introduce new actions not present in $\mathcal{I}$, and the second one that $\mathcal{I}'$ has to include all the visible potential actions of $\mathcal{I}$. We lift $\sqsubseteq$ to assertions as follows:

$$I \sqsubseteq^{P} I' \stackrel{\text{def}}{=} \forall \iota.\ \langle\!|I|\!\rangle_{\iota} \sqsubseteq^{\{s \mid s,\iota \vDash_{\mathsf{SL}} P\}} \langle\!|I'|\!\rangle_{\iota}$$

We present a number of judgements that reduce action shifting and fencing conditions to logical entailments in Fig. 4.3 and 4.4. As with the local fencing judgements, the entailments in the premises of these rules are those of standard separation logic. The $\approx^{R}$ notation in the conclusion of some of the judgements denotes that shifting is valid both ways (*i.e.* $I_1 \approx^{R} I_2$ iff $I_1 \sqsubseteq^{R} I_2$ and $I_2 \sqsubseteq^{R} I_1$). We write $\mathsf{exact}\,(P)$ to denote that the assertion P is *exact*. That is, there exists l such that for all ι and l' where $l', \iota \vDash_{\mathsf{SL}} P$ then $l = l'$.

The premises of the form $p \cap q \vdash_{\mathsf{SL}} \mathsf{emp}$ assert that the states described by p and q *do not intersect*; that is, for all $\iota \in \mathsf{LEnv}$, and for all l_1, l_2 such that $l_1, \iota \vDash_{\mathsf{SL}} p$ and $l_2, \iota \vDash_{\mathsf{SL}} q$, then:

$$\forall l \in \mathsf{LState}.\, l \leq l_1 \wedge l \leq l_2 \implies l = \mathbf{0}_{\mathsf{L}}$$

The $p \cap q \vdash_{\mathsf{SL}} \mathsf{emp}$ entailment can be rewritten equivalently as follows:

$$p \cap q \vdash_{\mathsf{SL}} \mathsf{emp} \Leftrightarrow p \vdash_{\mathsf{SL}} \neg\,(\mathit{true} * (\neg\mathsf{emp} \wedge (\mathit{true} \mathrel{-\!\!\circledast} q)))$$

4.2 Programming Language and Proof System

We define the CoLoSL proof system for deriving local Hoare triples for a simple concurrent imperative programming language. The proof system and

$$\frac{P \vdash Q \quad I \sqsubseteq^{Q} I'}{I \sqsubseteq^{P} I'} \qquad \frac{f \rhd I \cup I_1 \quad I_1 \sqsubseteq^{f} I_2}{I \cup I_1 \sqsubseteq^{f} I \cup I_2} \qquad \frac{I \vdash_{\mathrm{I}} I' \quad I' \sqsubseteq^{f} \emptyset}{I \sqsubseteq^{f} \emptyset}$$

$$\overline{\bigcup_{i \in I} \{\mathcal{K} : \exists \bar{y}. P_i \rightsquigarrow Q\} \approx^{true} \left\{\mathcal{K} : \exists \bar{y}. \bigvee_{i \in I} P_i \rightsquigarrow Q\right\}}$$

$$\overline{\bigcup_{i \in I} \{\mathcal{K} : \exists \bar{y}. P \rightsquigarrow Q_i\} \approx^{true} \left\{\mathcal{K} : \exists \bar{y}. P \rightsquigarrow \bigvee_{i \in I} Q_i\right\}}$$

$$\overline{\left\{\mathcal{K} : \exists \overline{v_i \in S_i}^{i \in I}. P \rightsquigarrow Q\right\} \approx^{true} \bigcup_{\overline{w_i \in S_i}^{i \in I}} \left\{\mathcal{K} : P\overline{[w_i/v_i]}^{i \in I} \rightsquigarrow Q\overline{[w_i/v_i]}^{i \in I}\right\}}$$

$$\frac{\mathsf{exact}\,(r) \quad f \cap p \vdash_{\mathsf{SL}} \mathsf{emp}}{\{\mathcal{K} : p * r \rightsquigarrow q * r\} \sqsubseteq^{f} \emptyset} \qquad \frac{f \circledast p \vdash_{\mathsf{SL}} \mathit{false}}{\{\mathcal{K} : p \rightsquigarrow q\} \sqsubseteq^{f} \emptyset}$$

$$\frac{(p \mathrel{-\!\!\circledast} (p \circledast f)) * q \vdash_{\mathsf{SL}} \mathit{false}}{\{\mathcal{K} : p \rightsquigarrow q\} \sqsubseteq^{f} \emptyset}$$

$$\frac{f \circledast p \vdash_{\mathsf{SL}} \bigvee_{i \in I} f \circledast (p * r_i) \quad \mathsf{exact}\,(r_i) \ \text{ for } i \in I}{\{\mathcal{K} : p \rightsquigarrow q\} \ \approx^{f} \ \bigcup_{i \in I} \{\mathcal{K} : p * r_i \rightsquigarrow q * r_i\}}$$

Figure 4.4: Action shifting judgements; we write $I \approx^{f} I'$ for $I \sqsubseteq^{f} I' \wedge I' \sqsubseteq^{f} I$.

programming language of CoLoSL are defined as an instantiation of the views framework [5].

Programming Language The CoLoSL programming language is that of the views programming language [5] instantiated with a set of *atomic* commands; It consists of skip, sequential composition, branching, loops and parallel composition. The set of CoLoSL atomic commands comprises an *atomic* construct $\langle . \rangle$ enforcing an atomic behaviour when instantiated with any *sequential* command. The sequential commands of CoLoSL are composed of a set of *elementary* commands, skip, sequential composition, branching and loops excluding atomic constructs and parallel composition. That is, atomic commands cannot be nested or contain parallel composition. CoLoSL is parametric in the choice of elementary commands allowing for

suitable *instantiation* of CoLoSL depending on the programs being verified. For instance, in the token ring example of § 2 the set of elementary commands comprises variable lookup and assignment. We proceed with the formalisation of CoLoSL programming language.

Parameter 5 (Elementary commands)**.** Assume a set of elementary commands Elem, ranged over by $\mathbb{E}, \mathbb{E}_1, \cdots, \mathbb{E}_n$.

Parameter 6 (Elementary command axioms)**.** Given the separation algebra of machine states $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$, assume a set of axioms associated with elementary commands:

$$\mathrm{Ax}_{\mathsf{E}} : \mathcal{P}(\mathbb{M}) \times \mathsf{Elem} \times \mathcal{P}(\mathbb{M})$$

Definition 37 (Sequential commands)**.** Given the set of elementary commands Elem, the set of sequential commands Seqs, ranged over by $\mathbb{S}, \mathbb{S}_1, \cdots, \mathbb{S}_n$ is defined inductively as:

$$\mathbb{S} ::= \mathbb{E} \mid \text{ skip} \mid \mathbb{S}_1; \mathbb{S}_2 \mid \mathbb{S}_1 + \mathbb{S}_2 \mid \mathbb{S}^\star$$

Definition 38 (Sequential command axioms)**.** Given the axiomatisation of elementary commands Ax_{E}, the *axioms of sequential commands*:

$$\mathrm{Ax}_{\mathsf{S}} : \mathcal{P}(\mathbb{M}) \times \mathsf{Seqs} \times \mathcal{P}(\mathbb{M})$$

is defined as follows where we write $M, M', M'', \cdots$ to quantify over the elements of $\mathcal{P}(\mathbb{M})$.

$$\begin{array}{rcl}
\mathrm{Ax}_{\mathsf{S}} & \stackrel{\text{def}}{=} & \mathrm{Ax}_{\mathsf{E}} \cup A_{\text{skip}} \cup A_{Seq} \cup A_{Choice} \cup A_{Rec} \\
A_{\text{skip}} & \stackrel{\text{def}}{=} & \{(M, \text{ skip}, M) \mid M \in \mathcal{P}(\mathbb{M})\} \\
A_{Seq} & \stackrel{\text{def}}{=} & \left\{ (M, \mathbb{S}_1; \mathbb{S}_2, M') \;\middle|\; \begin{array}{l} (M, \mathbb{S}_1, M'') \in \mathrm{Ax}_{\mathsf{S}} \wedge \\ (M'', \mathbb{S}_2, M') \in \mathrm{Ax}_{\mathsf{S}} \end{array} \right\} \\
A_{Choice} & \stackrel{\text{def}}{=} & \left\{ (M, \mathbb{S}_1 + \mathbb{S}_2, M') \;\middle|\; \begin{array}{l} (M, \mathbb{S}_1, M') \in \mathrm{Ax}_{\mathsf{S}} \wedge \\ (M, \mathbb{S}_2, M') \in \mathrm{Ax}_{\mathsf{S}} \end{array} \right\} \\
A_{Rec} & \stackrel{\text{def}}{=} & \left\{ (M, \mathbb{S}^\star, M) \;\middle|\; (M, \mathbb{S}, M) \in \mathrm{Ax}_{\mathsf{S}} \right\}
\end{array}$$

Definition 39 (Atomic commands)**.** Given the set of sequential commands Seqs, the set of*atomic commands* Atom, ranged over by $\mathbb{A}, \mathbb{A}_1, \cdots, \mathbb{A}_n$ is:

$$\mathbb{A} ::= \langle \mathbb{S} \rangle$$

Definition 40 (Atomic command axioms)**.** Given the axioms of sequential commands Ax_{S}, the *axioms of atomic commands*:

$$\mathrm{Ax}_{\mathsf{A}} : \mathcal{P}\left(\mathsf{World}\right) \times \mathsf{Atom} \times \mathcal{P}\left(\mathsf{World}\right)$$

is defined as follows where we write $W, W', \cdots$ to quantify over the elements of $\mathcal{P}\left(\mathsf{World}\right)$.

$$\mathrm{Ax}_{\mathsf{A}} \stackrel{\text{def}}{=} \left\{ \; (W, \langle \mathbb{S} \rangle, W') \;\middle|\; \begin{array}{l} (M_1, \mathbb{S}, M_2) \in \mathrm{Ax}_{\mathsf{S}} \wedge \\ W \Rrightarrow^{\{M_1\}\{M_2\}} W' \end{array} \right\}$$

Definition 41 (Programming language)**.** Given the set of atomic commands Atom, the set of CoLoSL *commands* Comm, ranged over by $\mathbb{C}, \mathbb{C}_1, \cdots \mathbb{C}_n$, is defined by the following grammar.

$$\mathbb{C}_{:} := \mathbb{A} \mid \texttt{skip} \mid \mathbb{C}_1; \mathbb{C}_2 \mid \mathbb{C}_1 + \mathbb{C}_2 \mid \mathbb{C}_1 \parallel \mathbb{C}_2 \mid \mathbb{C}^{\star}$$

Proof Rules Our proof rules are of the form $\vdash \{P\}\ \mathbb{C}\ \{Q\}$ and carry an implicit assumption that the pre- and post-conditions of their judgements are stable. Since we build CoLoSL as an instantiation of the views framework [5], our proof rules correspond to those of [5] with the atomic commands axiomatised as per Def. 40.

Definition 42 (Proof rules)**.** The *proof rules* of CoLoSL are as described below.

$$\frac{}{\vdash \{P\}\ \mathrm{skip}\ \{P\}}\ \textsc{Skip} \qquad \frac{\forall \iota.\ (\llbracket P \rrbracket_\iota,\ \mathbb{A},\ \llbracket Q \rrbracket_\iota) \in \mathrm{Ax}_{\mathsf{A}}}{\vdash \{P\}\ \mathbb{A}\ \{Q\}}\ \textsc{Atom}$$

$$\frac{\vdash \{P\}\mathbb{C}_1\{R\} \quad \vdash \{R\}\mathbb{C}_2\{Q\}}{\vdash \{P\}\ \mathbb{C}_1; \mathbb{C}_2\ \{Q\}}\ \textsc{Seq} \qquad \frac{\vdash \{P_1\}\mathbb{C}_1\{Q_1\} \quad \vdash \{P_2\}\mathbb{C}_2\{Q_2\}}{\vdash \{P_1 * P_2\}\ \mathbb{C}_1 \parallel \mathbb{C}_2\ \{Q_1 * Q_2\}}\ \textsc{Par}$$

$$\frac{\vdash \{P\}\ \mathbb{C}\ \{Q\}}{\vdash \{P * R\}\, \mathbb{C}\, \{Q * R\}}\ \textsc{Frame} \qquad \frac{P \Rrightarrow P' \quad \vdash \{P'\}\, \mathbb{C}\, \{Q'\} \quad Q' \Rrightarrow Q}{\vdash \{P\}\ \mathbb{C}\ \{Q\}}\ \textsc{Conseq}$$

$$\frac{\vdash \{P\}\mathbb{C}_1\{Q\} \quad \vdash \{P\}\mathbb{C}_2\{Q\}}{\vdash \{P\}\ \ \mathbb{C}_1 + \mathbb{C}_2\ \ \{Q\}}\ \textsc{Choice} \qquad \frac{\vdash \{P\}\, \mathbb{C}\, \{P\}}{\vdash \{P\}\ \ \mathbb{C}^{\star}\ \ \{P\}}\ \textsc{Rec}$$

Most proof rules are standard from disjoint concurrent separation logic [14]. In the Conseq judgement, $\Rrightarrow$ denotes the repartitioning of the state as described in Def. 31.

4.3 Operational Semantics

We define the operational semantics of CoLoSL in terms of a set of *concrete* (low-level) states. Recall that the states of CoLoSL, namely worlds, are parametric in the separation algebra of machine states $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$. As such, we require that the choice of concrete states is also parametrised in CoLoSL. Since CoLoSL is an instantiation of the views framework, its operational semantics is as defined in [5] provided with the set of concrete states and the *semantics of atomic commands*. The semantics of atomic commands are defined in terms of the *interpretation of sequential and elementary commands*. Finally, as CoLoSL can be instantiated with any set of elementary commands Elem, the interpretation of elementary commands is also a parameter in CoLoSL. We proceed with the formalisation of the ingredients necessary for defining the operational semantics of atomic commands.

Parameter 7 (Concrete states)**.** Assume a set of concrete states $\mathcal{S}$ ranged over by $\sigma, \sigma_1, \cdots, \sigma_n$.

Parameter 8 (Elementary command interpretation)**.** Given the set of concrete states $\mathcal{S}$, assume an *elementary interpretation function* associating each elementary command with a non-deterministic state transformer:

$$[\![.]\!]_{\mathsf{E}}(.) : \mathsf{Elem} \to \mathcal{S} \to \mathcal{P}(\mathcal{S})$$

We lift the interpretation function to a set of concrete states such that for $S \in \mathcal{P}(\mathcal{S})$:

$$[\![\mathbb{E}]\!]_{\mathsf{E}}(S) \stackrel{\text{def}}{=} \bigcup_{\sigma \in S} ([\![\mathbb{E}]\!]_{\mathsf{E}}(\sigma))$$

Definition 43 (Sequential command interpretation)**.** Given the interpretation function of elementary commands $[\![.]\!]_{\mathsf{E}}(.)$, the *interpretation function for sequential commands*:

$$[\![.]\!]_{\mathsf{S}}(.) : \mathsf{Seqs} \to \mathcal{S} \to \mathcal{P}(\mathcal{S})$$

is defined inductively over the structure of sequential commands as follows:

$$\begin{array}{rcl}
[\![\mathbb{E}]\!]_{\mathsf{S}}(\sigma) & \stackrel{\text{def}}{=} & [\![\mathbb{E}]\!]_{\mathsf{E}}(\sigma) \\
[\![\,\mathrm{skip}]\!]_{\mathsf{S}}(\sigma) & \stackrel{\text{def}}{=} & \{\sigma\} \\
[\![\mathbb{S}_1; \mathbb{S}_2]\!]_{\mathsf{S}}(\sigma) & \stackrel{\text{def}}{=} & \{\sigma_2 \mid S = [\![\mathbb{S}_1]\!]_{\mathsf{S}}(\sigma) \ \wedge \ \sigma_2 \in [\![\mathbb{S}_2]\!]_{\mathsf{S}}(S)\} \\
[\![\mathbb{S}_1 + \mathbb{S}_2]\!]_{\mathsf{S}}(\sigma) & \stackrel{\text{def}}{=} & [\![\mathbb{S}_1]\!]_{\mathsf{S}}(\sigma) \cup [\![\mathbb{S}_2]\!]_{\mathsf{S}}(\sigma) \\
[\![\mathbb{S}^\star]\!]_{\mathsf{S}}(\sigma) & \stackrel{\text{def}}{=} & [\![\,\mathrm{skip} + \mathbb{S}; \mathbb{S}^\star]\!]_{\mathsf{S}}(\sigma)
\end{array}$$

where we lift the interpretation function to a set of concrete states such that for $S \in \mathcal{P}(\mathcal{S})$:

$$[\![\mathbb{S}]\!]_{\mathsf{S}}(S) \stackrel{\text{def}}{=} \bigcup_{\sigma \in S} ([\![\mathbb{S}]\!]_{\mathsf{S}}(\sigma))$$

Definition 44 (Atomic Command Interpretation)**.** Given the sequential command interpretation function $[\![.]\!]_{\mathsf{S}}(.)$, the *interpretation function for atomic commands*:

$$[\![.]\!]_{\mathsf{A}}(.) : \mathsf{Atom} \to \mathcal{S} \to \mathcal{P}(\mathcal{S})$$

is defined as :

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}}(\sigma) \stackrel{\text{def}}{=} [\![\mathbb{S}]\!]_{\mathsf{S}}(\sigma)$$

We lift the interpretation function to a set of concrete states such that for $S \in \mathcal{P}(\mathcal{S})$:

$$[\![\mathbb{S}]\!]_{\mathsf{A}}(S) \stackrel{\text{def}}{=} \bigcup_{\sigma \in S} ([\![\mathbb{S}]\!]_{\mathsf{A}}(\sigma))$$

4.4 Soundness

In order to establish the soundness of CoLoSL program logic, we relate its proof judgements to its operational semantics. To this end, we relate the CoLoSL states, *i.e.* worlds, to the concrete states by means of a *reification function*. The reification of worlds is defined in terms of relating (reifying) machine states in $\mathbb{M}$ to concrete states in $\mathcal{S}$. As such, given that CoLoSL is parametric in the choice of underlying machine states, the *reification of machine states* is also a parameter in CoLoSL.

Since CoLoSL is an instantiation of the views framework [5], its soundness follows immediately from the soundness of views, provided that the atomic axioms are sound with respect to their operational semantics. Recall that the axioms of atomic commands are defined in terms of the axioms pertaining to elementary commands which are parametrised in CoLoSL. Thus, to ensure the soundness of atomic commands, we require that the elementary axioms are also sound with respect to their operational semantics. We proceed with

Parameter 9 (Machine state reification)**.** Given the separation algebra of machine states $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$, assume a *reification function* $\lfloor . \rfloor_{\mathbb{M}} : \mathbb{M} \to \mathcal{P}(\mathcal{S})$, relating machine states to sets of concrete states.

Parameter 10 (Elementary soundness)**.** Given the separation algebra of machine states $(\mathbb{M}, \bullet_{\mathbb{M}}, \mathbf{0}_{\mathbb{M}})$,and their reification function $\lfloor . \rfloor_{\mathbb{M}}$, assume for

each elementary command $\mathbb{E} \in \mathsf{Elem}$, its axiom $(M_1, \mathbb{E}, M_2) \in \mathrm{Ax}_{\mathsf{E}}$ and any given machine state $m \in \mathbb{M}$ the following *soundness* property holds.

$$[\![\mathbb{E}]\!]_{\mathsf{E}} (\lfloor M_1 \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \subseteq \lfloor M_2 \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}$$

Definition 45 (Reification)**.** The *reification of worlds*, $\lfloor . \rfloor_W : \mathsf{World} \to \mathcal{P}(\mathcal{S})$ is defined as:

$$\lfloor (l, g, \mathcal{I}) \rfloor_W \stackrel{\text{def}}{=} \lfloor (l \circ g)_{\mathsf{M}} \rfloor_{\mathbb{M}}$$

Theorem 1 (Atomic command soundness)**.** For all $\mathbb{A} \in \mathsf{Atom}$, $(W_1, \mathbb{A}, W_2) \in \mathrm{Ax}_{\mathsf{A}}$ and $w \in \mathsf{World}$:

$$[\![\mathbb{A}]\!]_{\mathsf{A}} (\lfloor W_1 \bullet \{w\} \rfloor_W) \subseteq \lfloor W_2 \bullet_{\mathbb{M}} R(\{w\}) \rfloor_W$$

Proof. By induction over the structure of $\mathbb{A}$.

Case $\langle \mathbb{S} \rangle$
Pick an arbitrary $\mathbb{S} \in \mathsf{Seqs}$, $w \in \mathsf{World}$ and $W_1, W_2 \in \mathcal{P}(\mathsf{World})$ such that $(W_1, \langle \mathbb{S} \rangle, W_2) \in \mathrm{Ax}_{\mathsf{A}}$. From the definition of Ax_{A} we then know there exists $M_1, M_2 \in \mathcal{P}(\mathbb{M})$ such that:

$$(M_1, \mathbb{S}, M_2) \in \mathrm{Ax}_{\mathsf{S}} \wedge W_1 \Rrightarrow^{\{M_1\}\{M_2\}} W_2 \tag{4.27}$$

RTS.

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} (\lfloor W_1 \bullet \{w\} \rfloor_W) \subseteq \lfloor W_2 \bullet R(\{w\}) \rfloor_W$$

Proof. Pick an arbitrary $w_1 = (l, g, \mathcal{I}) \in W_1$; it then suffices to show that there exists $w_2 \in W_2$ and $w' \in R(w)$ such that

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} (\lfloor w_1 \bullet w \rfloor_W) = \lfloor w_2 \bullet w' \rfloor_W \tag{4.28}$$

Given $w_1 = (l_1, g_1, \mathcal{I}_1)$, from the definitions of $[\![.]\!]_{\mathsf{A}}(.)$ and $\lfloor . \rfloor_W$, and the properties of $\bullet$ and $\bullet_{\mathbb{M}}$ we have:

$$\begin{aligned} [\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} (\lfloor w_1 \bullet w \rfloor_W) &= [\![\mathbb{S}]\!]_{\mathsf{S}} (\lfloor w_1 \bullet w \rfloor_W) \\ &= [\![\mathbb{S}]\!]_{\mathsf{S}} \left(\lfloor (l_1 \circ g_1)_{\mathsf{M}} \bullet_{\mathbb{M}} (w_{\mathsf{L}})_{\mathsf{M}} \rfloor_{\mathbb{M}} \right) \end{aligned} \tag{4.29}$$

On the other hand, from (4.27) and the definition of $\Rrightarrow$ we know there exists $m_1 \in M_1$ and $m' \in \mathbb{M}$ such that

$$m_1 \circ m' = (l_1 \circ g_1)_{\mathsf{M}} \wedge \tag{4.30}$$

$$\begin{aligned} &\forall m_2 \in M_2 . \exists w_2 = (l_2, g_2, \mathcal{I}_2) \in W_2 . \\ &\quad m_2 \circ m' = (l_2 \circ g_2)_{\mathsf{M}} \wedge (w_1, w_2) \in G \end{aligned} \tag{4.31}$$

Consequently from (4.29) and (4.30) we have:

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} \left(\lfloor w_1 \bullet w \rfloor_W \right) = [\![\mathbb{S}]\!]_{\mathsf{S}} \left(\left\lfloor m_1 \bullet_{\mathbb{M}} m' \bullet_{\mathbb{M}} (w_{\mathsf{L}})_{\mathsf{M}} \right\rfloor_{\mathbb{M}} \right) \tag{4.32}$$

From (4.27) and Lemma 13 we can rewrite (4.32) as

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} \left(\lfloor w_1 \bullet w \rfloor_W \right) \subseteq \left\lfloor M_2 \bullet_{\mathbb{M}} \{ m' \bullet_{\mathbb{M}} (w_{\mathsf{L}})_{\mathsf{M}} \} \right\rfloor_{\mathbb{M}}$$

That is, there exits $m_2 \in M_2$ such that

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} \left(\lfloor w_1 \bullet w \rfloor_W \right) = \left\lfloor m_2 \bullet_{\mathbb{M}} m' \bullet_{\mathbb{M}} (w_{\mathsf{L}})_{\mathsf{M}} \right\rfloor_{\mathbb{M}} \tag{4.33}$$

From (4.31) we know there exists $w_2 \in \mathsf{World}$ such that

$$w_2 = (l_2, g_2, \mathcal{I}_2) \in W_2 \wedge m_2 \circ m' = (l_2 \circ g_2)_{\mathsf{M}} \tag{4.34}$$

$$(w_1, w_2) \in G \tag{4.35}$$

From the definition of $\lfloor . \rfloor_W$ and the properties of $\bullet_{\mathbb{M}}$ and $\bullet$ we can thus rewrite (4.33) as

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} \left(\lfloor w_1 \bullet w \rfloor_W \right) = \lfloor (l_2 \circ w_{\mathsf{L}}, g_2, \mathcal{I}_2) \rfloor_W \tag{4.36}$$

From (4.35) and Lemma 14 we know there exists $w' \in \mathsf{World}$ such that

$$w' = (w_{\mathsf{L}}, g_2, \mathcal{I}_2) \wedge w' \in R(w) \tag{4.37}$$

Consequently, from (4.34), (4.36) and (4.37) we know there exists $w_2 \in W_2$ and $w' \in R(w)$ such that

$$[\![\langle \mathbb{S} \rangle]\!]_{\mathsf{A}} \left(\lfloor w_1 \bullet w \rfloor_W \right) = \left\lfloor w_2 \bullet w' \right\rfloor_W$$

as required.

□

5. Examples

5.1 Concurrent Spanning Tree

Programs manipulating arbitrary graphs present the following significant challenge for compositional verification: because of deep sharing between different components of a graph, changes to one subgraph may affect other subgraphs which may point into it. This makes it hard to reason about updates to each subgraph in isolation. In a concurrent setting, this difficulty compounds with the fact that threads working on different parts of the graph may affect each other in ways that are difficult to reason about locally to each subgraph. Central to those issues is the fact that different subgraphs present *unspecified sharing* and may overlap in arbitrary ways. We now demonstrate on a concurrent spanning tree example that CoLoSL may be just the tool you need in this case, as it naturally deals with arbitrary overlapping views of the shared state, and allows one to tailor interferences to a given subjective view.

Our example, presented in Fig. 5.4, operates on a *directed binary graph* (henceforth simply *graph*), i.e. a directed graph where each node has at most two successors, called its left and right children. The program computes a in-place spanning tree of the graph, i.e. a tree that covers all nodes of the graphs from a given root, concurrently as follows: each time a new node is encountered, two new threads are created that each prune the edges of the left and right children recursively. A mark bit is associated to each node to keep track of which ones have already been visited. Each thread returns whether the root of the subgraph it was operating on had already been visited; if so, the parent thread removes the link from its root node to the corresponding child. Intuitively, it is allowed to do so because, since the child was marked by another thread, the child has to be already reachable via some other path in the graph.

We will prove that, given a shared graph as input, the program always returns a tree, i.e. all sharing and cycles have been appropriately removed.

Pleasingly, the CoLoSL specification achieves maximal locality and allows us to reason only on the current subgraph manipulated by a thread, instead of the whole graph. Because of arbitrary sharing between the two, a global specification would be unpleasant indeed! However, we do not establish that the final tree indeed spans the original graph. The reason it does is subtle indeed, as are the invariants required, but in ways unrelated to our main issue, which is to provide tight specifications for each thread.

To reason about this program, following Hobor and Villard [10], we use two representations of graphs. The first is a mathematical representation $\gamma = (V, E)$ where V is a finite set of vertices and $E : V \to (V \uplus \{\text{ null}\}) \times (V \uplus \{\text{ null}\})$ is a function associating each vertex with at most two successors, where null denotes the absence of an edge from the node. We write $n \in \gamma$ for $n \in V$, $\gamma(n)$ for $E(n)$ and $|\gamma|$ for $|V|$.

Mathematical graphs are connected to a second, in-memory representation by an inductive predicate $\mathsf{graph}\,(x, \gamma)$, denoting a spatial (in-heap) graph rooted at address x corresponding to the mathematical graph γ. The predicate definition uses the overlapping conjunction to account for the sharing between the left and right children and for potential cycles in the graph, as shown in Fig. 5.2. The basic action we allow on spatial graphs is to *mark* a node, changing its mark field from 0 to 1 and claiming ownership of its left and right pointers in the process. Such an action is allowed by a *marking* capability of the form $\mathsf{a_m}\,(n, e)$ where n denotes the vertex (address) and e the edge via which vertex n is visited. For instance, the capabilities associated with marking of vertex z in Fig. 5.1 are $\mathsf{a_m}\,(z, y.r)$ and $\mathsf{a_m}\,(z, w.l)$. Note that the parameterisation of our actions are merely a notational convenience and can be substituted for their full definitions. Given a graph at root address x, in order to account for the ability to mark the root vertex x, we introduce a logical (virtual) root edge $\mathcal{R}$ into x as depicted in Fig. 5.1 together with its associated marking capability $\mathsf{a_m}\,(x, \mathcal{R})$. The shared state contains node x which can be either unmarked ($\mathsf{U}\,(x, l, r)$) or marked ($\mathsf{M}\,(x)$); as well as the left and right subgraphs captured recursively by $\mathsf{G}\,(l, \gamma)$ and $\mathsf{G}\,(r, \gamma)$.

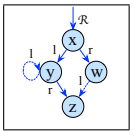

Figure 5.1: A graph.

Each vertex is represented as three consecutive cells in the heap tracking the mark bit and the addresses of the left (l) and right (r) subgraphs. For brevity, we write $x \mapsto m, l, r$ for $x \mapsto m * x + 1 \mapsto l * x + 2 \mapsto r$, and $x.m$, $x.l$, and $x.r$ for x, $x + 1$, and $x + 2$, respectively. When vertex x is in the unmarked state, the whole cell $x \mapsto 0, l, r$ and the capabilities to mark the children reside in the shared state. In the marked state, the shared state only

$$\mathsf{graph}\,(x,\gamma) \stackrel{\text{def}}{=} [\mathsf{a_m}\,(x,\mathcal{R})] * \boxed{\mathsf{G}\,(x,\gamma)}_{I_\gamma} \qquad I_\gamma \stackrel{\text{def}}{=} \bigcup_{n\in\gamma} I(n)$$
$$\mathsf{G}\,(x,\gamma) \stackrel{\text{def}}{=} (x = \text{ null} \wedge \mathsf{emp}) \vee x \in \gamma \wedge \exists l, r.\, \gamma(x) = (l,r) \wedge (\mathsf{U}\,(x,l,r) \vee \mathsf{M}\,(x)) \circledast \mathsf{G}\,(l,\gamma) \circledast \mathsf{G}\,(r,\gamma)$$
$$\mathsf{U}\,(x,l,r) \stackrel{\text{def}}{=} x \mapsto 0,l,r * [\mathsf{a_m}\,(l,x.l)] * [\mathsf{a_m}\,(r,x.r)]$$
$$\mathsf{M}\,(x) \stackrel{\text{def}}{=} x \mapsto 1$$
$$I(n) \stackrel{\text{def}}{=} \{\mathsf{a_m}\,(n,-) : \exists l, r.\, \mathsf{U}\,(n,l,r) \rightsquigarrow \mathsf{M}\,(n)\}$$

Figure 5.2: Global specification of the graph predicate.

$$\mathsf{g}\,(x,\gamma) \stackrel{\text{def}}{=} (x = \text{ null} \wedge \mathsf{emp}) \vee x \in \gamma \wedge \exists l, r.\, \gamma(x) = (l,r) \wedge \boxed{\mathsf{U}\,(x,l,r) \vee \mathsf{M}\,(x)}_{I(x)} * \mathsf{g}\,(l,\gamma) * \mathsf{g}\,(r,\gamma)$$
$$\mathsf{t}\,(x,\gamma) \stackrel{\text{def}}{=} (x = \text{ null} \wedge \mathsf{emp}) \vee x \in \gamma \wedge \exists l, r.\, \gamma(x) = (l,r) \wedge \boxed{\mathsf{M}\,(x)}_{I(x)} * \exists l' \in \{l, \text{ null}\}.\, \exists r' \in \{r, \text{ null}\}.\, [\mathsf{a_m}\,(l,x.l)] * x.l \mapsto l' * \mathsf{t}\,(l',\gamma) * [\mathsf{a_m}\,(r,x.r)] * x.r \mapsto r' * \mathsf{t}\,(r',\gamma)$$

Figure 5.3: Local specification of the graph predicate.

contains $x.m \mapsto 1$: the left and right subgraphs (pointers and capabilities) have been claimed by the thread who marked the node, while other threads need not access the children of x once they see that x is already marked. The atomic CAS instruction prevents several threads to concurrently mark the same node and claim ownership of the same resource.

The interference associated with the graph is described as the union of interferences pertaining to the vertices of the graph ($n \in \gamma$). For each vertex $n \in \gamma$, the only permitted action is that of marking n which can be carried out by any of the marking capabilities associated with node n ($\mathsf{a_m}\,(n,-)$). Note that the anonymous quantification $-$ is yet another notational shorthand and can be substituted for the following more verbose definition.

$$I(n) \stackrel{\text{def}}{=} \bigcup_{p\in\gamma} \left(\bigcup_{e\in\{p.l,p.r,\mathcal{R}\}} \mathsf{a_m}\,(n,e) : \exists l, r.\, \mathsf{U}\,(n,l,r) \rightsquigarrow \mathsf{M}\,(n) \right)$$

Fig. 5.4 shows an in place concurrent algorithm for calculating a spanning tree of a graph. The $\mathsf{graph}\,(x,\gamma)$ predicate defined in Fig. 5.2 is a *global* account of the graph in that it captures all vertices and the interference associated with them. However, our spanning tree algorithm operates *locally* as it is called upon recursively for each node. That is, for each span(n)

call (where $\mathrm{n} \mapsto n$ and $n \in \gamma$), the footprint of the call is limited to node n. Moreover, in order to reason about the concurrent recursive calls $\mathrm{span(x.l)} \parallel \mathrm{span(x.r)}$, we need to *split* the state into two $\star$-composed states prior to the calls, pass each constituent state onto the relevant thread and combine the resulting states by $\star$ composition through an application of the PAR rule. We thus provide a *local* specification of the graph, $\mathsf{g}(x,\gamma)$ as defined in Fig. 5.3 such that for all $n, p \in \gamma$ and $e \in \{p.l, p.r, \mathcal{R}\}$

$$\left\{ \mathrm{n} \mapsto n \star \mathrm{b} \mapsto - \star [\mathsf{a_m}(n,e)] \star \mathsf{g}(n,\gamma) \right\}$$
$$\mathrm{b{:=}\ span(n)}$$
$$\left\{ \mathrm{n} \mapsto n \star [\mathsf{a_m}(n,e)] \star (\mathrm{b} \mapsto 1 \star \mathsf{t}(n,\gamma) \vee \mathrm{b} \mapsto 0 \star \mathsf{t}(\mathrm{null},\gamma)) \right\}$$

The definition of the $\mathsf{g}(x,\gamma)$ predicate is similar to that of $\boxed{\mathsf{G}(x,\gamma)}_{I_\gamma}$ except that the global view $\boxed{\mathsf{G}(x,\gamma)}_{I_\gamma}$ that describes the resources associated with all $|\gamma|$ vertices has been replaced by $|\gamma|$ $\star$-composed more local views, each describing the resources of a vertex $n \in \gamma$. Moreover, the interference of each local view concerning a vertex $n \in \gamma$ has been shifted from I_γ to $I(n)$ as to reflect only those actions that affect n.

Similarly, the $\mathsf{t}(x,\gamma)$ predicate represents a *tree* rooted at x, as is standard in separation logic [15], and consists of $|\gamma|$ subjective views one for each vertex in γ. The assertion of each subjective view reflects that the corresponding vertex (x) has been marked $\boxed{\mathsf{M}(x)}_{I(x)}$. The resources associated with each node x, namely the left and right pointers and the corresponding marking capabilities have been claimed by the marking thread and moved into the local state. The vertex addressed by the left pointer of x (i.e. l') corresponds to either the initial value prior to marking (l where $\gamma(x) = (l,r)$) or null when l has more than one predecessors and has been marked by another thread, making the whole predicate stable against actions of the program and the environment.

We now demonstrate how to obtain the local specification $\mathsf{g}(x,\gamma)$ from the global specification of Fig. 5.2. When expanding the definition of $\mathsf{G}(x,\gamma)$, there are two cases to consider depending on whether or not $x = \mathrm{null}$. In what follows we only consider the case where $x \neq \mathrm{null}$ since the derivation in the case of $x = \mathrm{null}$ is trivial. Let P and Q predicates be defined as below.

$$P \stackrel{\text{def}}{=} \circledast_{n\in\gamma} (\gamma(n) = (l,r) \wedge (\mathsf{U}(n,l,r) \vee \mathsf{M}(n)))$$
$$Q \stackrel{\text{def}}{=} \circledast_{n\in\gamma} \left(\gamma(n) = (l,r) \wedge \boxed{\mathsf{U}(n,l,r) \vee \mathsf{M}(n)}_{I(n)}\right)$$

From the definitions of $\mathsf{G}(x,\gamma)$ and $\mathsf{g}(x,\gamma)$ we then have:

$$\mathsf{G}(x,\gamma) \iff P \qquad\qquad \mathsf{g}(x,\gamma) \iff Q$$

In order to derive the local specification $\mathsf{g}\,(x, \gamma)$ from the global specification $\mathsf{G}\,(x, \gamma)$, it thus suffices to show $\boxed{P}_{I_\gamma} \Rrightarrow Q$ as demonstrated below.

$$
\begin{aligned}
\boxed{P}_{I_\gamma} &\overset{(\textsc{Copy})}{\Longrightarrow} \underbrace{\boxed{P}_{I_\gamma} * \cdots * \boxed{P}_{I_\gamma}}_{|\gamma| \text{ times}} \\
&\overset{(\textsc{Forget})}{\Longrightarrow} \circledast_{n \in \gamma} \left(\gamma(n) = (l, r) \wedge \boxed{\mathsf{U}\,(n, l, r) \vee \mathsf{M}\,(n)}_{I_\gamma} \right) \\
&\overset{(\textsc{Shift})}{\Rrightarrow} \circledast_{n \in \gamma} \left(\gamma(n) = (l, r) \wedge \boxed{\mathsf{U}\,(n, l, r) \vee \mathsf{M}\,(n)}_{I(n)} \right) \\
&\overset{\text{def}}{\Longleftrightarrow} Q
\end{aligned}
$$

$//\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * \mathsf{graph}\,(x,\gamma)$

$//\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * \mathsf{a_m}\,(x,\mathcal{R}) * \boxed{\mathsf{G}\,(x,\gamma)}_{I_\gamma}\}$

$//\{\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \mathsf{g}\,(x,\gamma)\}$

b:= span(x) {

$//\left\{\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \exists l, r. \boxed{\mathsf{U}\,(x,l,r) \vee \mathsf{M}\,(x)}_{I(x)} * \mathsf{g}\,(l,\gamma) * \mathsf{g}\,(r,\gamma)\right\}$

res:= ⟨ CAS(x.m, 0, 1) ⟩;

$//\left\{\begin{array}{l}\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \boxed{\mathsf{M}\,(x)}_{I(x)} * \exists l, r.\, \mathsf{g}\,(l,\gamma) * \mathsf{g}\,(r,\gamma) \\ * \left(\mathrm{res} \mapsto 0 \vee \left(\ \mathrm{res} \mapsto 1 * x.l \mapsto l * x.r \mapsto r * [\mathsf{a_m}\,(l,x.l)] * [\mathsf{a_m}\,(r,x.r)]\ \right)\right)\end{array}\right\}$

if (res) **then** {

$//\left\{\begin{array}{l}\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \boxed{\mathsf{M}\,(x)}_{I(x)} * \mathrm{res} \mapsto 1 \\ * \exists l, r.\, x.l \mapsto l * x.r \mapsto r * [\mathsf{a_m}\,(l,x.l)] * \mathsf{g}\,(l,\gamma) * [\mathsf{a_m}\,(r,x.r)] * \mathsf{g}\,(r,\gamma)\end{array}\right\}$

$//\{[\mathsf{a_m}\,(l,x.l)] * \mathsf{g}\,(l,\gamma) * [\mathsf{a_m}\,(r,x.r)] * \mathsf{g}\,(r,\gamma)\}$

b1:= span(x.l) ‖ b2:= span(x.r)

$//\left\{\begin{array}{l}[\mathsf{a_m}\,(l,x.l)] * ((\mathrm{b1} \mapsto 1 * \mathsf{t}\,(l,\gamma)) \vee \mathrm{b1} \mapsto 0) * \\ [\mathsf{a_m}\,(r,x.r)] * ((\mathrm{b2} \mapsto 1 * \mathsf{t}\,(r,\gamma)) \vee \mathrm{b2} \mapsto 0)\end{array}\right\}$

$//\left\{\begin{array}{l}\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \boxed{\mathsf{M}\,(x)}_{I(x)} * \mathrm{res} \mapsto 1 \\ * \exists l, r.\, x.l \mapsto l * [\mathsf{a_m}\,(l,x.l)] * ((\mathrm{b1} \mapsto 1 * \mathsf{t}\,(l,\gamma)) \vee \mathrm{b1} \mapsto 0) * \\ \qquad x.r \mapsto r * [\mathsf{a_m}\,(r,x.r)] * ((\mathrm{b2} \mapsto 1 * \mathsf{t}\,(r,\gamma)) \vee \mathrm{b2} \mapsto 0)\end{array}\right\}$

if (!b1) then

[x.l]:= null

if (!b2) then

[x.r]:= null

$//\left\{\begin{array}{l}\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \boxed{\mathsf{M}\,(x)}_{I(x)} * \mathrm{res} \mapsto 1 * \mathrm{b1} \mapsto - * \mathrm{b2} \mapsto - \\ * \exists l, r.\, \exists l' \in \{l, \mathrm{null}\}.\ [\mathsf{a_m}\,(l,x.l)] * x.l \mapsto l' * \mathsf{t}\,(l',\gamma) * \\ \qquad \exists r' \in \{r, \mathrm{null}\}.\ [\mathsf{a_m}\,(r,x.r)] * x.r \mapsto r' * \mathsf{t}\,(r',\gamma)\end{array}\right\}$

$//\{\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * \mathrm{res} \mapsto 1 * \mathrm{b1} \mapsto - * \mathrm{b2} \mapsto - * \mathsf{a_m}\,(x,\mathcal{R}) * \mathsf{t}\,(x,\gamma)\}$

}

$//\left\{\begin{array}{l}\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * \\ (\mathrm{res} \mapsto 1 * \mathsf{t}\,(x,\gamma)) \vee \left(\mathrm{res} \mapsto 0 * \boxed{\mathsf{M}\,(x)}_{I(x)} * \mathsf{g}\,(l,\gamma) * \mathsf{g}\,(r,\gamma)\right)\end{array}\right\}$

$//\{\mathrm{x} \mapsto x * \mathrm{b} \mapsto - * [\mathsf{a_m}\,(x,\mathcal{R})] * (\mathrm{res} \mapsto 1 * \mathsf{t}\,(x,\gamma)) \vee (\mathrm{res} \mapsto 0)\}$

return res

}

$//\{\mathrm{x} \mapsto x * [\mathsf{a_m}\,(x,\mathcal{R})] * ((\mathrm{b} \mapsto 1 * \mathsf{t}\,(x,\gamma)) \vee \mathrm{b} \mapsto 0)\}$

Figure 5.4: Concurrent Spanning Tree Implementation

5.2 Set Module

In this section we consider a concurrent set module as described in [6]. We first produce the set specification in CoLoSL and show how to reason about its operations and verify them with respect to their specification. We then compare the CoLoSL specification of the set module against its specification in Concurrent Abstract Predicates (CAP) described in [6]. We demonstrate that our CoLoSL reasoning considerably improves on CAP by producing a *more concise* specification and allowing for *more local* reasoning.

CoLoSL Specification

We implement a set as a sorted singly-linked list with no duplicate elements (since it represents a set) and one lock per node as described in [6]. Our set module provides three operations: contains(h,v), add(h,v) and remove(h,v). As the name suggests, the contains(h,v) function checks whether value v is contained in the set at h; the add(h,v) and remove(h,v) functions add/remove value v to/from the list, respectively.

Fig. 5.9-5.7 illustrate a possible implementation of the set operations. All three operations proceed by traversing the sorted list from the head address and locating the first node in the list holding a value v' greater than or equal to v (through the locate procedure). The algorithm for locating such a node begins by locking the head node; it then moves down the list by hand-over-hand locking whereby first the node following the one currently held is locked and subsequently the previously locked node is released. No thread can access a node locked by another thread or traverse past it. As such, no thread can overtake an other thread in accessing the list.

Fig. 5.5 shows a specification of the set module in CoLoSL similar to that of Concurrent Abstract Predicates (CAP) defined in [6]. In what follows we first give a description of the predicates of Fig. 5.5 and subsequently contrast our specification with that of CAP.

Since CoLoSL is parametric in the separation algebra of capabilities and their assertions, we instantiate it with a *fractional heap* to represent the separation algebra of capabilities and write heap-like assertions of the form $a \overset{\pi}{\Rightarrow} b$ to denote capability a with permission π. Moreover, our capabilities are *stateful* in that they can capture some additional information (b). As we demonstrate below, this leads to conciser specifications. For readability we write $a \Rightarrow b$ for $a \overset{1}{\Rightarrow} b$, $a \overset{\pi}{\Rightarrow} -$ for $\exists b.\, a \overset{\pi}{\Rightarrow} b$ and $a \Rightarrow -$ for $\exists b.\, a \overset{1}{\Rightarrow} b$.

$$\mathsf{in}\,(h,v) \stackrel{\text{def}}{=} \exists\pi.\,\mathsf{isLock}\,(h,\pi) * h.\mathsf{V} \Rrightarrow - * \mathsf{L}_{\in}\,(h,v)$$
$$\mathsf{out}\,(h,v) \stackrel{\text{def}}{=} \exists\pi.\,\mathsf{isLock}\,(h,\pi) * h.\mathsf{V} \Rrightarrow - * \mathsf{L}_{\notin}\,(h,v)$$

$$\mathsf{L}_{\in}\,(h,v) \stackrel{\text{def}}{=} \exists L.\,\mathsf{sorted}\,(L) \wedge v \in L \wedge \mathsf{lsg}\,(h,\,\mathrm{null},-\infty :: L + +\{\infty\},h)$$
$$\mathsf{L}_{\notin}\,(h,v) \stackrel{\text{def}}{=} \exists L.\,\mathsf{sorted}\,(L) \wedge v \notin L \wedge \mathsf{lsg}\,(h,\,\mathrm{null},-\infty :: L + +\{\infty\},h)$$

$$\mathsf{lsg}\,(x,z,L,h) \stackrel{\text{def}}{=} (L = [] \wedge x = z \wedge \mathsf{emp}) \vee \exists y,v,L'.\,L = v :: L' \wedge \boxed{\mathsf{node}\,(x,v,y)}_{I(h,x)} * \mathsf{lsg}\,(y,z,L',h)$$

$$\mathsf{node}\,(x,v,y) \stackrel{\text{def}}{=} \mathsf{U}\,(x,v,y) \vee \mathsf{L}\,(x,v,y)$$
$$\mathsf{U}\,(x,v,y) \stackrel{\text{def}}{=} \mathsf{link}\,(x,y) * \mathsf{val}\,(x,v,y)$$
$$\mathsf{L}\,(x,v,y) \stackrel{\text{def}}{=} \mathsf{locked}\,(x) * \mathsf{val}\,(x,v,y)$$

$$\mathsf{link}\,(x,y) \stackrel{\text{def}}{=} x.next \mapsto y * \mathsf{isLock}\,(y,1) * x.\mathsf{M} \Rrightarrow -$$
$$\mathsf{val}\,(x,v,y) \stackrel{\text{def}}{=} x.value \mapsto v * x.\mathsf{N} \Rrightarrow y$$
$$\mathsf{isLock}\,(x,\pi) \stackrel{\text{def}}{=} x.\mathsf{L} \stackrel{\pi}{\Rrightarrow} - * \boxed{x.lock \mapsto 0 * x.\mathsf{U} \Rrightarrow - \vee x.lock \mapsto 1}_{L(x)}$$
$$\mathsf{locked}\,(x) \stackrel{\text{def}}{=} x.\mathsf{U} \Rrightarrow - * \boxed{x.lock \mapsto 1}_{L(x)}$$

$$L(x) \stackrel{\text{def}}{=} \left\{ \begin{array}{r} x.\mathsf{L} \stackrel{-}{\Rrightarrow} - : \left\{ x.lock \mapsto 0 * x.\mathsf{U} \Rrightarrow - \rightsquigarrow x.lock \mapsto 1 \right\} \\ x.\mathsf{U} \Rrightarrow - : \left\{ x.lock \mapsto 1 \rightsquigarrow x.lock \mapsto 0 * x.\mathsf{U} \Rrightarrow - \right\} \\ x.\mathsf{L} \Rrightarrow - * x.\mathsf{U} \Rrightarrow - : \left\{ x.lock \mapsto 1 \rightsquigarrow x.\mathsf{L} \stackrel{1}{\Rrightarrow} - * x.\mathsf{U} \stackrel{1}{\Rrightarrow} - \right\} \end{array} \right\}$$

$$I(h,x) \stackrel{\text{def}}{=} \left\{ \begin{array}{r} true : \left\{ \begin{array}{l} \exists v,y.\,\mathsf{U}\,(x,v,y) \rightsquigarrow \mathsf{L}\,(x,v,y) \\ \exists v,y.\,\mathsf{L}\,(x,v,y) \rightsquigarrow \mathsf{U}\,(x,v,y) \end{array} \right\} \\ x.\mathsf{M} \Rrightarrow - * h.\mathsf{V} \Rrightarrow - : \\ \left(\begin{array}{ll} \exists v',y,z. & \mathsf{L}\,(x,v',y) * \mathsf{L}\,(y,v,z) \\ & \quad \rightsquigarrow \mathsf{L}\,(x,v',z) * \mathsf{L}\,(y,v,z) \\ \exists v',w,y. & v' < v \wedge \mathsf{L}\,(w,v',y) * \mathsf{L}\,(x,v,y) \\ & \quad \rightsquigarrow \mathsf{L}\,(w,v',y) \\ \exists v_1,v_2,y,z,w. & v_1 < v < v_2 \wedge \mathsf{L}\,(x,v_1,y) * \mathsf{val}\,(w,v,y) * \mathsf{val}\,(y,v_2,z) \\ & \quad \rightsquigarrow \mathsf{L}\,(x,v_1,w) * \mathsf{val}\,(w,v,y) * \mathsf{val}\,(y,v_2,z) \end{array} \right) \end{array} \right\}$$

Figure 5.5: CoLoSL specification of the concurrent set module.

isLock(x, π) / locked(x) Every node in the singly-linked list is protected by a lock that is to be acquired prior to its modification. The $\mathsf{isLock}\,(x, \pi)$ predicate is similar to that in [6] and states that the lock at address x can be acquired with permission $\pi \in (0, 1]$. Multiple threads may attempt to acquire the lock at once and thus we use the π argument to reflect this sharing where 1 denotes exclusive right to acquisition while $\pi \in (0, 1)$ accounts for sharing of the lock between multiple threads. The $\mathsf{isLock}\,(x, \pi)$ predicate asserts that the thread's local state contains the capability $x.\mathsf{L} \overset{\pi}{\Rightarrow} -$ to acquire the lock and that the lock resides in the shared state where at any one point either the lock is unlocked ($x.lock \mapsto 0$) and the shared state contains the capability to unlock it ($x.\mathsf{U} \Rightarrow -$); or it is locked ($x.lock \mapsto 1$) and the unlocking capability has been claimed by the locking thread.

The $\mathsf{locked}\,(x)$ predicate asserts that the thread's local state contains the exclusive capability to unlock the lock ($[x.\mathsf{U} \Rightarrow -]$); and that the x is in the locked state ($x.lock \mapsto 1$).

L(x) denotes the interference associated with the lock on node at address x. When a thread holds a non-zero locking capability on node x ($x.\mathsf{L} \overset{-}{\Rightarrow} -$), it can change the lock state from unlocked (0) to locked (1) and claim the exclusive unlocking capability in doing so ($x.\mathsf{U} \Rightarrow -$). Similarly, a thread holding the full unlocking capability on x ($x.\mathsf{U} \Rightarrow -$) can change the lock state from locked to unlocked and return the full unlocking capability to the shared state. When removing a value v from the set, the node containing the value v must be removed from the shared state and disposed of. However, recall that CoLoSL does not have an explicit mechanism for unsharing resources modelling a dual behaviour to that of the EXTEND principle. Instead, resources can be unshared through actions explicitly specified as part of their associated interference. This is the case in the last component of $L(x)$; it is concerned with *unsharing* of node x's lock and making it a local resource. It states that when a thread holds full permission on both locking and unlocking capabilities of x, it can claim its lock in exchange for the capabilities.

in(h, v) / out(h, v) The $\mathsf{in}\,(h, v)$ predicate states that the set at head address h contains value v and captures exclusive right to alter the set with respect to v by changing whether v belongs to it. It asserts that the thread owns some capability (π) to acquire the lock associated with head address ($\mathsf{isLock}\,(h, \pi)$) and that the thread owns the exclusive capability to alter the set with respect to value v ($h.\mathsf{V} \Rightarrow -$) should it own the capability to modify the address at which value v is stored (*cf.* the description of

$\mathsf{U}\,(x, v, y)$ predicate). The underlying singly-inked list is captured by the $\mathsf{L}_{\in}\,(h, v)$ predicate. The $\mathsf{L}_{\in}\,(h, v)$ predicate uses an auxiliary carrier sorted list L (such that $v \in L$) to capture the contents of the singly-linked list via the lsg predicate. For simpler implementation, we extend L with two sentinel values $-\infty$ and ∞, one at each end to avoid corner cases such as removing the first/last element of the list. The $\mathsf{out}\,(h, v)$ predicate is analogous to $\mathsf{in}\,(h, v)$ and corresponds to the case where the set does not contain v.

$\mathsf{lsg}(x,\ z,\ L,\ h)$ This predicate is defined inductively and describes a *segment* of the list at h that starts at address x and extends upto (but not including) address z and contains the elements in the mathematical list L. When L is empty, it corresponds to a void segment and yields no resources (emp); otherwise, it is defined as the composition of the first node of the segment at address x, $\boxed{\mathsf{node}\,(x, v, y)}_{I(h,x)}$, and the tail of the list segment ($\mathsf{lsg}\,(y, z, L', h)$). The $\mathsf{node}\,(x, v, y)$ predicate describes a node at address x with value v and successor y and can be either unlocked ($\mathsf{U}\,(x, v, y)$) or locked ($\mathsf{L}\,(x, v, y)$).

$\mathsf{U}(x,\ v,\ y)$ / $\mathsf{L}(x,\ v,\ y)$ The $\mathsf{U}\,(x, v, y)$ predicate states that the node at address x is unlocked, it contains value v and comes before the node at address y. The statement of the $\mathsf{L}\,(x, v, y)$ predicate is analogous and describes the node when locked. A thread may modify the next pointer of node at address x, only when it holds the modification capability $x.\mathsf{M} \Rrightarrow -$ in its local state. When the node is in the unlocked state, no thread can modify it and thus the modification capability $x.\mathsf{M} \Rrightarrow -$ as well as the next pointer of the node ($x.next \mapsto y$) lie in the shared state. Recall that when traversing the list by hand-over-hand locking, no thread can overtake an other in traversing the list. Consequently, a node can only be locked by a thread that has already acquired the lock associated with the previous node. Therefore, when the node x is in the unlocked state, the exclusive capability to lock its successor at address y ($\mathsf{isLock}\,(y, 1)$) also lies in the shared state. When a thread successfully locks the node at x (and is hence in possession of the $\mathsf{locked}\,(x)$ resource), it can then claim the next pointer, the modification capability and the locking capability pertaining to its successors (captured by $\mathsf{link}\,(x, y)$), and move them into its local state. In return, it transfers the $\mathsf{locked}\,(x)$ resource to the shared state as evidence that it is indeed the locking thread.

In both locked and unlocked states, the shared state contains the value field ($x.value \mapsto v$) and the "successor" capability $x.\mathsf{N} \Rrightarrow y$. This capability

is used to track the current successor of x even when the node is locked and its next pointer has been claimed by the locking thread.

***I*(*h*, *x*)** As discussed above, given a node at address x with successor y, any thread in possession of the $\mathsf{locked}\,(x)$ resource may change its state from unlocked ($\mathsf{U}\,(x, v, y)$) to locked ($\mathsf{L}\,(x, v, y)$) and claim the $\mathsf{link}\,(x, y)$ resource in exchange for $\mathsf{locked}\,(x)$. This is captured by the first action associated with the *true* assertion; that is, no additional capability is required for performing this action and holding the $\mathsf{locked}\,(x)$ resource locally is sufficient for performing the action. Conversely, as described by the second action of *true*, a thread in possession of the $\mathsf{link}\,(x, y)$ resource may change its state from locked ($\mathsf{L}\,(x, v, y)$) to unlocked ($\mathsf{U}\,(x, v, y)$) and return $\mathsf{link}\,(x, y)$ to the shared state in return for $\mathsf{locked}\,(x)$.

Consider a node at address y, with value v, successor z and predecessor x. A thread may remove value v from the set provided that both the node and its predecessor are locked by the thread; and the thread holds the value capability $h.\mathsf{V} \Rrightarrow -$ as well as the update capabilities pertaining to both the node itself ($x.\mathsf{M} \Rrightarrow -$) and its predecessor node ($y.\mathsf{M} \Rrightarrow -$). This is captured by the first two actions associated with $x.\mathsf{M} \Rrightarrow - * h.\mathsf{V} \Rrightarrow -$ in $I(h, x)$. First, the next pointer of x is redirected to z (the first action of $x.\mathsf{M} \Rrightarrow - * h.\mathsf{V} \Rrightarrow -$); and then node y is removed from the shared state (the second action of $y.\mathsf{M} \Rrightarrow - * h.\mathsf{V} \Rrightarrow -$). Similarly, for a thread to insert the value v in between node x and node y, the predecessor node x must be locked; and the thread should hold the value capability $h.\mathsf{V} \Rrightarrow -$ as well as the update capability on x. This is reflected in the last action of $x.\mathsf{M} \Rrightarrow - * h.\mathsf{V} \Rrightarrow -$.

Reasoning about Set Operations

Fig. 5.6-5.9 illustrate the implementation of the set operations along with an outline of the reasoning steps involved in establishing their correctness against their CoLoSL specification. In what follows, we give an account of reasoning about the set add operation as outlined in Fig. 5.7; we demonstrate that given a set at address h, where value v may or may not be present in the set, after a call to the add(h,v) operation the set will contain the value v. That is, the add(h,v) operation satisfies the following specification.

$$\{\mathsf{in}\,(\mathrm{h}, v) \vee \mathsf{out}\,(\mathrm{h}, v)\}$$
$$\mathrm{add(h},v)$$
$$\{\mathsf{in}\,(\mathrm{h}, v)\}$$

The implementation of add proceeds by locating the index at which the new node is to be inserted through a call to the locate(h,v) operation. The locate(h,v) call traverses the list by hand-over-hand locking until it locates the addresses of node p with value v_p and node c with value v_c such that $v_p < v \leq v_c$. It then locks p and claims its next pointer (link(p, c)) as allowed by L(p). Note that since the list contains the sentinel values $-\infty$ and $+\infty$ at either end, for any given value v' the locate operation always finds v_1 and v_2 such that $v_1 < v' \leq v_2$. If c.value $=v$ and consequently the set contains value v, node p is unlocked and the operating thread simply returns. On the other hand if $v_c > v$, a new node z with value v and successor c is allocated and the shared state is extended by the resources associated with the new node z as follows. First, the shared state is extended by z's lock and in doing so the locking and unlocking capabilities pertaining to z are generated on the fly. That is the z.lock $\mapsto 0$ predicate is replaced by isLock$(z, 1)$ as shown in the following derivation.

$$\left\{\begin{array}{l}\exists L_1, L_2, v_p, v_c, \pi.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathsf{lsg}\,(\mathrm{c},\ \mathrm{null}, v_c :: L_2, \mathrm{h})\end{array}\right\}$$

(frame off) $\{\mathsf{emp}\}$

alloc+assign $\Rrightarrow$ $\{\mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c} * \mathrm{z.lock} \mapsto 0\}$

Extend $\Rrightarrow$ $\left\{\mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c} * \mathrm{z}.\mathsf{L} \Rrightarrow - * \boxed{\mathrm{z.lock} \mapsto 0 * \mathrm{z}.\mathsf{U} \Rrightarrow -}_{L(\mathrm{z})}\right\}$

isLock def. $\{\mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c} * \mathsf{isLock}\,(\mathrm{z}, 1)\}$

(frame on)
$$\left\{\begin{array}{l}\exists L_1, L_2, v_p, v_c, \pi.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathsf{lsg}\,(\mathrm{c},\ \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * \mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c} * \mathsf{isLock}\,(\mathrm{z}, 1)\end{array}\right\}$$

The shared state is then extended with the remaining resources of node z and the relevant capabilities are also generated; subsequently, the subjective views of nodes p, z and c are combined through an application of the Merge principle as demonstrated by the following derivation.

$$\left\{\begin{array}{l}\exists L_1, L_2, v_p, v_c, \pi.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathsf{lsg}\,(\mathrm{c},\ \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * \mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c} * \mathsf{isLock}\,(\mathrm{z}, 1)\end{array}\right\}$$

(frame off) $\{\mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathrm{z.value} \mapsto v * \mathrm{z.next} \mapsto \mathrm{c}\}$

link def. $\Longrightarrow$ $\{\mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p}.\mathsf{M} \Rrightarrow - * \mathsf{isLock}\,(\mathrm{c}, 1) * \mathrm{z.next} \mapsto \mathrm{c} * \mathrm{z.value} \mapsto v\}$

Extend $\Rrightarrow$
$$\left\{\mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p}.\mathsf{M} \Rrightarrow - * \boxed{\begin{array}{l}\mathsf{isLock}\,(\mathrm{c}, 1) * \mathrm{z.next} \mapsto \mathrm{c} * \mathrm{z}.\mathsf{M} \Rrightarrow - \\ * \mathrm{z.value} \mapsto v * \mathrm{z}.\mathsf{N} \Rrightarrow \mathrm{c}\end{array}}_{I(\mathrm{z})}\right\}$$

$$\xRightarrow{\mathsf{U}\ \text{def.}} \left\{ \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \boxed{\mathsf{U}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} \right\}$$

$$\xRightarrow{\mathsf{node}\ \text{def.}} \left\{ \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} \right\}$$

$$\xRightarrow{(\text{frame on})} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{c},\ \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} * \mathsf{isLock}\,(\mathrm{z}, 1) \end{array} \right\}$$

$$\xRightarrow{\mathsf{lsg}\ \text{def.}} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \boxed{\mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d})}_{I(\mathrm{c})} \\ * \mathsf{lsg}\,(\mathrm{d},\ \mathrm{null}, L_2, \mathrm{h}) * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} * \mathsf{isLock}\,(\mathrm{z}, 1) \end{array} \right\}$$

$$\xRightarrow{\textsc{Merge}\times 2} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{d},\ \mathrm{null}, L_2, \mathrm{h}) * \mathsf{isLock}\,(\mathrm{z}, 1) \end{array} \right\}$$

At this point, since the locking thread holds the next pointer of p in its local state ($\mathrm{p.next} \mapsto \mathrm{c}$), it modifies it to point to the new node z; and through the action of $\mathrm{p.M} \Rrightarrow - * \mathrm{h.V} \Rrightarrow -$, the $\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})$ predicate is updated as $\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{z})$. Recall that the $\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})$ predicate states that the node at p is locked, it holds value v_p and prior to locking its successor was the node at address c. As such, since the value of the new node z lies between that of p and c ($v_p < v < v_c$), this action allows us to insert z in between p and c provided that z itself points to c. This is captured by the following derivation where (†) denotes the application of the action associated with the $\mathrm{p.M} \Rrightarrow - * \mathrm{h.V} \Rrightarrow -$ capability in $I(\mathrm{p})$.

$$\left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \mathrm{p.next} \mapsto \mathrm{c} * \mathrm{p.M} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{d},\ \mathrm{null}, L_2, \mathrm{h}) * \mathsf{isLock}\,(\mathrm{z}, 1) \end{array} \right\}$$

$$\xRightarrow{(\mathrm{p.next} := \mathrm{z})} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \mathrm{p.next} \mapsto \mathrm{z} * \mathrm{p.M} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{d},\ \mathrm{null}, L_2, \mathrm{h}) * \mathsf{isLock}\,(\mathrm{z}, 1) \end{array} \right\}$$

$$\xRightarrow{\mathsf{link}\ \text{def.}} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 + +\{v_p\} + +v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h.V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{link}\,(\mathrm{p}, \mathrm{z}) \end{array} \right\}$$

$$\overset{(\dagger)}{\Rrightarrow} \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 ++ \{v_p\} ++ v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{link}\,(\mathrm{p}, \mathrm{z}) \end{array} \right\}$$

All that remains is to unlock the node at p; to do this, first the state of the node is changed from locked ($\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{z})$) to unlocked ($\mathsf{U}\,(\mathrm{p}, v_p, \mathrm{z})$) by applying the action of $\mathrm{p}.\mathsf{M} \Rrightarrow -$ capability. Subsequently, through several applications of Copy, Forget and Shift principles, the subjective views of p, z and c nodes are recovered as follows where (†) denotes the application of the action associated with the $\mathrm{p}.\mathsf{M} \Rrightarrow -$ capability in $I(\mathrm{p})$.

$$\left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 ++ \{v_p\} ++ v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{L}\,(\mathrm{c}, v_c, \mathrm{d}) *}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{link}\,(\mathrm{p}, \mathrm{z}) \end{array} \right\}$$

$$\text{(frame off)} \quad \left\{ \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{link}\,(\mathrm{p}, \mathrm{z}) \right\}$$

$$\overset{(\dagger)}{\Rrightarrow} \quad \left\{ \boxed{\mathsf{U}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{locked}\,(\mathrm{p}) \right\}$$

$$\overset{\mathsf{node}\text{ def.}}{\Longrightarrow} \quad \left\{ \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{locked}\,(\mathrm{p}) \right\}$$

$$\overset{\text{Copy}\times 2}{\Longrightarrow} \quad \left\{ \begin{array}{l} \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z}) * \mathsf{node}\,(\mathrm{z}, v, \mathrm{c}) * \mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \mathsf{locked}\,(\mathrm{p}) \end{array} \right\}$$

$$\overset{\text{Forget}\times 3}{\Longrightarrow} \quad \left\{ \begin{array}{l} \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z})}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} \\ * \boxed{\mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{p}) \cup I(\mathrm{z}) \cup I(\mathrm{c})} * \mathsf{locked}\,(\mathrm{p}) \end{array} \right\}$$

$$\overset{\text{Shift}\times 3}{\Rrightarrow} \quad \left\{ \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z})}_{I(\mathrm{p})} * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} * \boxed{\mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{c})} * \mathsf{locked}\,(\mathrm{p}) \right\}$$

$$\text{(frame on)} \quad \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi, d.\, v_p < v < v_c \wedge \mathsf{sorted}\,(L_1 ++ \{v_p\} ++ v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{z})}_{I(\mathrm{p})} * \boxed{\mathsf{node}\,(\mathrm{z}, v, \mathrm{c})}_{I(\mathrm{z})} * \boxed{\mathsf{node}\,(\mathrm{c}, v_c, d)}_{I(\mathrm{c})} * \mathsf{locked}\,(\mathrm{p}) \end{array} \right\}$$

$$\overset{\mathsf{lsg}\text{ def.}}{\Longrightarrow} \left\{ \begin{array}{l} \exists L, \pi.\, v \in L \wedge \mathsf{sorted}\,(L) \wedge \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{null}, L, \mathrm{h}) \\ * \mathsf{locked}\,(\mathrm{p}) \end{array} \right\}$$

$$\overset{\mathsf{in}\text{ def.}}{\Longrightarrow} \left\{ \mathsf{in}\,(\mathrm{h}, v) * \mathsf{locked}\,(\mathrm{p}) \right\}$$

Finally, p's lock is released by applying the action of $\mathrm{p}.\mathsf{U} \Rrightarrow -$ capability in $L(\mathrm{p})$ and we obtain $\mathsf{in}\,(\mathrm{h}, v)$ as required. This is demonstrated in the following derivation where (†) denotes applying the action of $\mathrm{p}.\mathsf{U} \Rrightarrow -$.

$$\left\{ \mathsf{in}\,(\mathrm{h}, v) * \mathsf{locked}\,(\mathrm{p}) \right\}$$

$$\overset{\text{locked def.}}{\Longrightarrow} \left\{ \mathsf{in}\,(\mathrm{h}, v) * \mathrm{p}.\mathsf{U} \Rrightarrow - * \boxed{\mathrm{p.lock} \mapsto 1}_{L(\mathrm{p})} \right\}$$
$$\overset{(\dagger)}{\Rrightarrow} \left\{ \mathsf{in}\,(\mathrm{h}, v) * \boxed{\mathrm{p.lock} \mapsto 0 * \mathrm{p}.\mathsf{U} \Rrightarrow -}_{L(\mathrm{p})} \right\}$$
$$\Longrightarrow \left\{ \mathsf{in}\,(\mathrm{h}, v) \right\}$$

Note that the final implication follows directly from the semantics of CoLoSL. That is, for all $P \in \mathsf{Assn}$ and $I \in \mathsf{IAssn}$, $\boxed{P}_I \implies \mathsf{emp}$ is valid.

$// \{\mathsf{in}\,(\mathrm{h}, v) \vee \mathsf{out}\,(\mathrm{h}, v)\}$
contains(h,v){
(p, c):= locate(h,v);
$$// \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi.\, v_p < v \leq v_c \wedge \mathsf{sorted}\,(L_1{+}{+}\{v_p\}{+}{+}v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathsf{lsg}\,(\mathrm{c}, \mathrm{null}, v_c :: L_2, \mathrm{h}) \end{array} \right\}$$
b:= (c.value ==v);
$$// \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi.\, v_p < v \wedge \mathsf{sorted}\,(L_1{+}{+}\{v_p\}{+}{+}v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{L}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{link}\,(\mathrm{p}, \mathrm{c}) * \mathsf{lsg}\,(\mathrm{c}, \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * ((v_c > v \wedge \mathrm{b} = \mathit{false}) \vee (v_c = v \wedge b = \mathit{true})) \end{array} \right\}$$
/* Apply the action of p.M $\Rrightarrow -$ wrapped in the definition of link (p, c). */
$$// \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi.\, v_p < v \wedge \mathsf{sorted}\,(L_1{+}{+}\{v_p\}{+}{+}v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{U}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{locked}\,(\mathrm{p}) * \mathsf{lsg}\,(\mathrm{c}, \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * ((v_c > v \wedge \mathrm{b} = \mathit{false}) \vee (v_c = v \wedge b = \mathit{true})) \end{array} \right\}$$
unlock(p);
$$// \left\{ \begin{array}{l} \exists L_1, L_2, v_p, v_c, \pi.\, v_p < v \leq v_c \wedge \mathsf{sorted}\,(L_1{+}{+}\{v_p\}{+}{+}v_c :: L_2) \wedge \\ \mathsf{isLock}\,(\mathrm{h}, \pi) * \mathrm{h}.\mathsf{V} \Rrightarrow - * \mathsf{lsg}\,(\mathrm{h}, \mathrm{p}, L_1, \mathrm{h}) \\ * \boxed{\mathsf{node}\,(\mathrm{p}, v_p, \mathrm{c})}_{I(\mathrm{p})} * \mathsf{lsg}\,(\mathrm{c}, \mathrm{null}, v_c :: L_2, \mathrm{h}) \\ * ((v_c > v \wedge \mathrm{b} = \mathit{false}) \vee (v_c = v \wedge b = \mathit{true})) \end{array} \right\}$$
$// \{((\mathsf{out}\,(\mathrm{h}, v) \wedge \mathrm{b} = \mathit{false}) \vee (\mathsf{in}\,(\mathrm{h}, v) \wedge b = \mathit{true}))\}$
}
$// \{((\mathsf{out}\,(\mathrm{h}, v) \wedge \mathrm{b} = \mathit{false}) \vee (\mathsf{in}\,(\mathrm{h}, v) \wedge b = \mathit{true}))\}$

Figure 5.6: Implementation of the set contains operation.

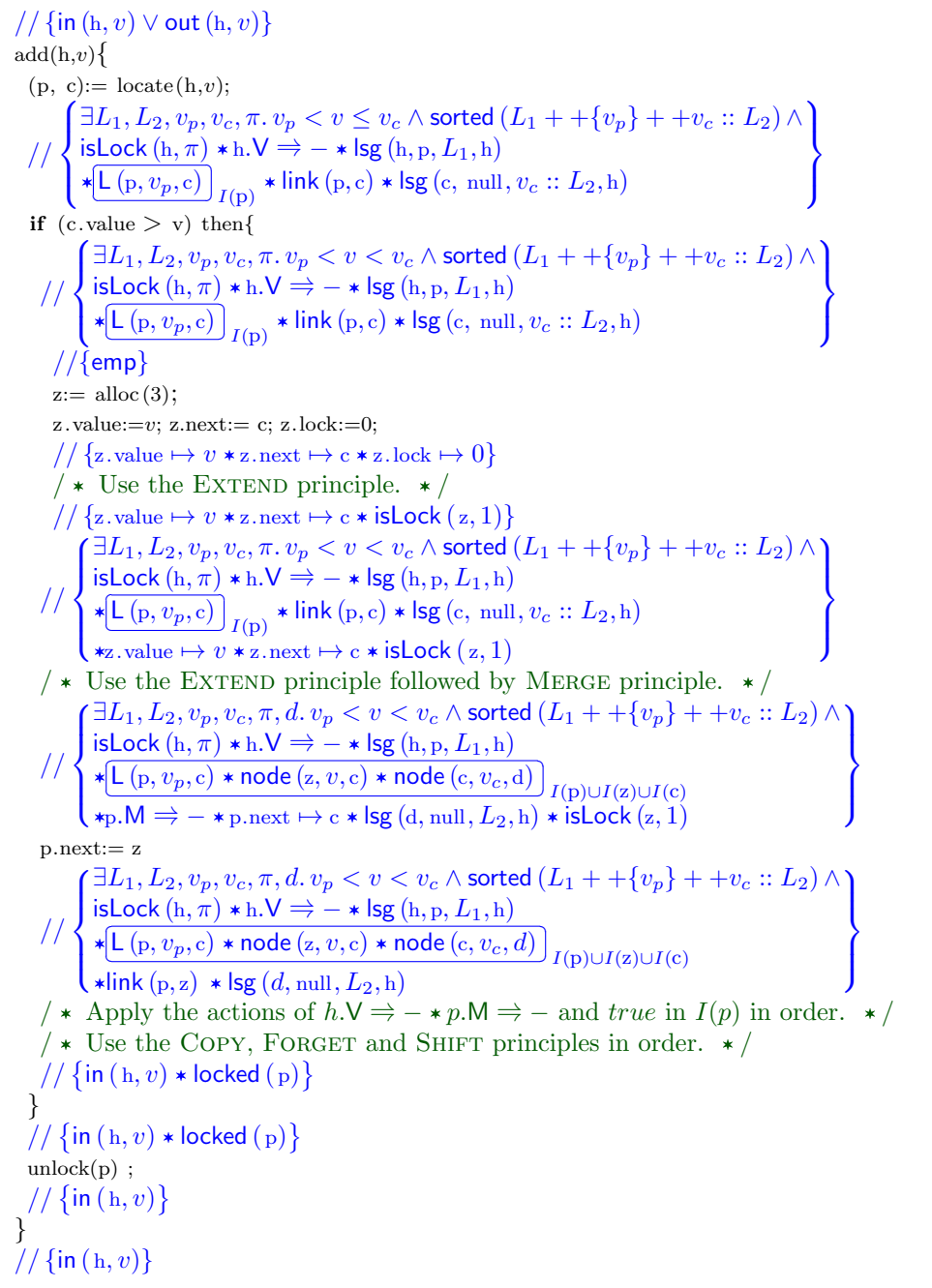

Figure 5.7: Implementation of the set add operation.

$//\,\{\mathsf{in}\,(\mathrm{h},v)\vee\mathsf{out}\,(\mathrm{h},v)\}$

remove(h,v){

(p, c):= locate(h,v);

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p,v_c,\pi.\,v_p<v\leq v_c\wedge\mathsf{sorted}\,(L_1++\{v_p\}++v_c::L_2)\wedge\\ \mathsf{isLock}\,(\mathrm{h},\pi)*\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})\\ *\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{c})}_{I(\mathrm{p})}*\mathsf{link}\,(\mathrm{p},\mathrm{c})*\mathsf{lsg}\,(\mathrm{c},\ \mathrm{null},v_c::L_2,\mathrm{h})\end{array}\right\}$$

if (c.value ==v) then{

lock(c.lock) ;

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,d,\pi.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++v::L_2)\wedge\\ \mathrm{h}.\mathsf{L}\overset{\pi}{\Rrightarrow}-*\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})\\ *\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{c})}_{I(\mathrm{p})}*\mathsf{link}\,(\mathrm{p},\mathrm{c})*\boxed{\mathsf{L}\,(\mathrm{c},v,d)}_{I(\mathrm{c})}*\mathsf{link}\,(\mathrm{c},d)*\mathsf{lsg}\,(d,\mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

z:= c.next ; [p.next]:= z ;

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\\ \mathrm{h}.\mathsf{L}\overset{\pi}{\Rrightarrow}-*\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})\\ *\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{c})}_{I(\mathrm{p})}*\mathrm{p}.\mathsf{M}\Rrightarrow-*\mathrm{p}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{c},1)\\ *\boxed{\mathsf{L}\,(\mathrm{c},v,\mathrm{z})}_{I(\mathrm{c})}*\mathrm{c}.\mathsf{M}\Rrightarrow-*\mathrm{c}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{z},1)*\mathsf{lsg}\,(\mathrm{z},\mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

/* Use the MERGE principle. */

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,\pi.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\\ \mathrm{h}.\mathsf{L}\overset{\pi}{\Rrightarrow}-*\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})\\ *\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{c})*\mathsf{L}\,(\mathrm{c},v,\mathrm{z})}_{I(\mathrm{p})\cup I(\mathrm{c})}*\mathrm{p}.\mathsf{M}\Rrightarrow-*\mathrm{p}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{c},1)\\ *\mathrm{c}.\mathsf{M}\Rrightarrow-*\mathrm{c}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{z},1)*\mathsf{lsg}\,(\mathrm{z},\ \mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

/* Apply the actions of $h.\mathsf{V}\Rrightarrow-*p.\mathsf{M}\Rrightarrow-$ cpability in $I(p)$ in order. */

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,\pi.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\\ \mathsf{isLock}\,(\mathrm{h},\pi)*\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})\\ *\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{z})}_{I(\mathrm{p})\cup I(\mathrm{c})}*\mathrm{p}.\mathsf{M}\Rrightarrow-*\mathrm{p}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{c},1)\\ *\mathsf{L}\,(\mathrm{c},v,\mathrm{z})*\mathrm{c}.\mathsf{M}\Rrightarrow-*\mathrm{c}.\mathrm{next}\mapsto\mathrm{z}*\mathsf{isLock}\,(\mathrm{z},1)*\mathsf{lsg}\,(\mathrm{z},\mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

/* Use SHIFT; apply the action of $c.\mathsf{L}\Rrightarrow-*c.\mathsf{U}\Rrightarrow-$ in $L(c)$. */

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,\pi.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\mathsf{isLock}\,(\mathrm{h},\pi)\\ *\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})*\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{z})}_{I(\mathrm{p})}*\mathsf{link}\,(\mathrm{p},\mathrm{z})*\mathsf{lsg}\,(\mathrm{z},\ \mathrm{null},L_2,\mathrm{h})\\ *\mathrm{c}.\mathrm{value}\mapsto v*\mathrm{c}.\mathrm{next}\mapsto\mathrm{z}*\mathrm{c}.\mathrm{lock}\mapsto1\end{array}\right\}$$

dispose(c, 3) ;

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,\pi.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\mathsf{isLock}\,(\mathrm{h},\pi)\\ *\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})*\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{z})}_{I(\mathrm{p})}*\mathsf{link}\,(\mathrm{p},\mathrm{z})*\mathsf{lsg}\,(\mathrm{z},\ \mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

}

$$//\left\{\begin{array}{l}\exists L_1,L_2,v_p<v,\pi,n.\,v\notin L_2\wedge\mathsf{sorted}\,(L_1++\{v_p\}++L_2)\wedge\mathsf{isLock}\,(\mathrm{h},\pi)\\ *\mathrm{h}.\mathsf{V}\Rrightarrow-*\mathsf{lsg}\,(\mathrm{h},\mathrm{p},L_1,\mathrm{h})*\boxed{\mathsf{L}\,(\mathrm{p},v_p,\mathrm{n})}_{I(\mathrm{p})}*\mathsf{link}\,(\mathrm{p},\mathrm{n})*\mathsf{lsg}\,(\mathrm{n},\ \mathrm{null},L_2,\mathrm{h})\end{array}\right\}$$

unlock(p); $//\,\{\mathsf{out}\,(\mathrm{h},v)\}$

}$//\,\{\mathsf{out}\,(\mathrm{h},v)\}$

Figure 5.8: Implementation of the set remove operation.

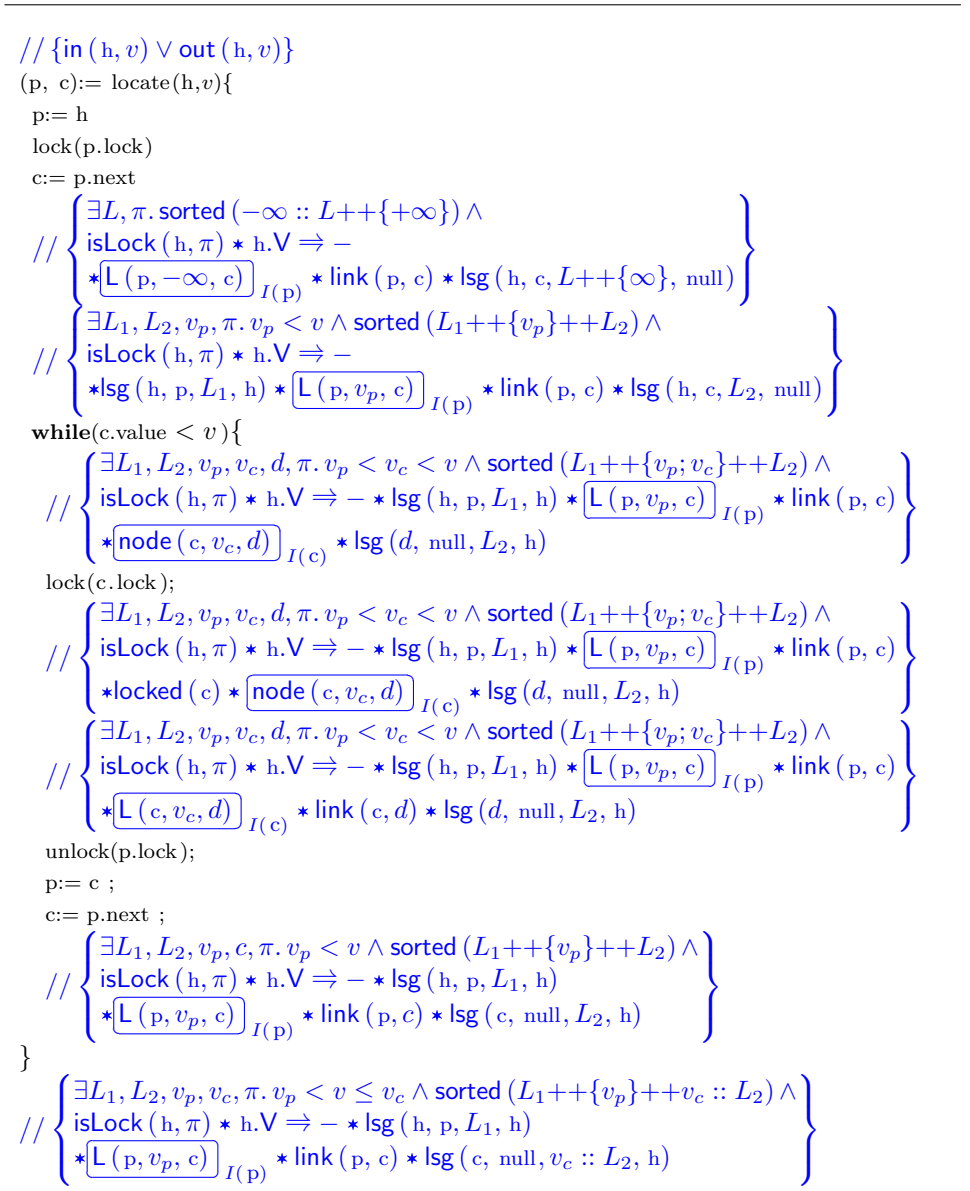

Figure 5.9: Implementation of the set locate operation.

CoLoSL vs. CAP Specification

The definitions of most of the predicates in Fig. 5.5 are analogous to the correspondingly-named CAP predicates specified in [6]. The difference however lies in the definitions of the $\mathsf{L}_{\in}$, $\mathsf{L}_{\notin}$, and lsg predicates. Fig. 5.10 presents the CAP definitions of these predicates.

There are two main advantages to the CoLoSL specification of the set module. In the CAP specification, all elements of the set (represented as a singly-linked list) reside in a *single* shared region labelled s. Each node at a given address x, is associated with an (*infinite*) set of update capabilities of the form $\text{LGAP}(x, y, v)$ for *all* possible addresses y and *all* possible values v. This is to capture all potential successor addresses y and all potential values v that may be stored at address x. For instance, since the value v' may be contained in the list at address a before address b, there must exist an update capability $\text{LGAP}(a, b, v')$ to accommodate possible modifications of value v' with respect to a and b. In order to modify a node, a thread can acquire the lock associated with the node and subsequently claim the relevant update capability. Since the capabilities associated with a region are *all* allocated at the point of region creation, the shared region is required to keep track of *all* (infinitely many) possible update capabilities $\text{LGAP}(x, y, v)$ associated with all addresses x, all possible successor addresses y and all values v. This is captured by the $\mathsf{gaps}\,(S, s)$ and $\mathsf{mygaps}\,(S, s)$ predicates through the infinite multiplicative star operator $\circledast$. The $\mathsf{gaps}\,(S, s)$ predicate uses an auxiliary mathematical set S to track those nodes of the list that are currently locked and thus infer which LGAP capabilities reside in the shared state and which ones have been removed. This results in an inevitably cluttered and a rather counter-intuitive specification since it seems unnatural to account for the capabilities pertaining to addresses not in the domain of the set.

On the other hand, in the CoLoSL specification we use a combination of the $x.\mathsf{M} \Rrightarrow -$ and $x.\mathsf{N} \Rrightarrow y$ capabilities to achieve the same effect for each address x. Since CoLoSL allows for dynamic extension of the shared state, the capabilities associated with each address x are generated upon *extension* of the shared state with those addresses, thus avoiding the need to account for capabilities concerning those addresses absent from the list.

Moreover, since the fractional heaps used to represent the separation algebra of capabilities are *stateful*, rather than having a distinct capability to modify the element at address x before address y for each possible successor address y, we appeal to a single capability of the form $x.\mathsf{N} \Rrightarrow y$ whereby the capability is modified accordingly to reflect the changes to the successor address. For instance, when the node at address x is redirected to point

$$\mathsf{L}_{\lhd}(h,v,s) \stackrel{\text{def}}{=} \exists L, S.\, \mathsf{sorted}(L) \wedge v \lhd L \wedge \mathsf{lsg}(h,\ \text{null}, S, -\infty :: L + +\{\infty\}) \\ * (\mathsf{gaps}(S,s) \wedge \mathsf{mygaps}(v,s)) \qquad \text{where } \lhd = \in \text{ or } \lhd = \notin$$

$$\mathsf{lsg}(x,z,S,L) \stackrel{\text{def}}{=} (L = [] \wedge S = \emptyset \wedge x = z \wedge \mathsf{emp}) \vee \\ (\exists y, v, L'.\, L = v :: L' \wedge \mathsf{U}(x,v,y) * \mathsf{lsg}(y,z,S,L')) \vee \\ \begin{pmatrix} \exists y, v, L', S'.\, L = v :: L' \wedge S = \{(x,y)\} \uplus S' \wedge \\ \mathsf{L}(x,v,y) * \mathsf{lsg}(y,z,S',L') \end{pmatrix}$$

$$\mathsf{gaps}(S,s) \stackrel{\text{def}}{=} \circledast(x,y) \notin S. \circledast v.\, [\textsc{LGap}(x,y,v)]_1^s * \\ \circledast(x,y) \in S. \exists w.\, \circledast v \neq w.\, [\textsc{LGap}(x,y,v)]_1^s \wedge \\ \forall x,y,w,z.\, (x,y) \in S \wedge (w,z) \in S \implies (x = w \iff y = z)$$

$$\mathsf{mygaps}(v,s) \stackrel{\text{def}}{=} \circledast x, y.\, [\textsc{LGap}(x,y,v)]_1^s * true$$

Figure 5.10: CAP predicates of the set module that contrast with the CoLoSL specification of Fig. 5.5.

from address y to a new location at address z, $x.\mathsf{N} \Rightarrow y$ is also updated to $x.\mathsf{N} \Rightarrow z$.

$$\mathsf{ring}\,(x,i,n) \stackrel{\text{def}}{=} \boxed{\exists v.\ \circledast_{k=0}^{i-1}\, x+k \mapsto v * \circledast_{k=i}^{n-1} x+k \mapsto v-1}\ \bigcup_{j=0}^{n-1} I'(j)$$

$$\mathsf{node}\,(x,i,n) \stackrel{\text{def}}{=} [\mathsf{a}_i] * \boxed{\begin{array}{l}\exists v.\ (x+a \mapsto v * x+b \mapsto v)\\ \vee\,(x+a \mapsto v+1 * x+b \mapsto v)\end{array}}_{I(i)}$$
where $a = min((i-1)\%n, i)$ and $b = max((i-1)\%n, i)$

$$\mathsf{turn}\,(x,i,n) \stackrel{\text{def}}{=} \begin{cases}\exists v.\, x \mapsto v * x+(n-1) \mapsto v & \text{if } i=0\\ \exists v.\, x+(i-1) \mapsto v+1 * x+i \mapsto v & \text{otherwise}\end{cases}$$

$$I(i) \stackrel{\text{def}}{=} I'(i) \cup I''(i)$$

$$I'(i) \stackrel{\text{def}}{=} \begin{cases}\left\{\begin{array}{ll}[\mathsf{a}_i]: & \exists v.\, x+(i-1) \mapsto v * x+i \mapsto v-1 \rightsquigarrow\\ & \quad x+(i-1) \mapsto v * x+i \mapsto v\end{array}\right\} & \text{if } i=0\\ \left\{\begin{array}{ll}[\mathsf{a}_0]: & \exists v.\, x \mapsto v * x+(n-1) \mapsto v \rightsquigarrow\\ & \quad x \mapsto v+1 * x+(n-1) \mapsto v\end{array}\right\} & \text{otherwise}\end{cases}$$

$$I''(i) \stackrel{\text{def}}{=}$$
$$\begin{cases}\left\{\begin{array}{l}[\mathsf{a}_{n-1}]: \exists v. x \mapsto v * x+(n-2) \mapsto v * x+(n-1) \mapsto v-1\\ \quad \rightsquigarrow x \mapsto v * x+(n-2) \mapsto v * x+(n-1) \mapsto v\end{array}\right\} & \text{if } i=0\\ \left\{\begin{array}{l}[\mathsf{a}_0]:\\ \exists v. x \mapsto v * x+1 \mapsto v * x+(n-1) \mapsto v-1\\ \quad \rightsquigarrow x \mapsto v+1 * x+1 \mapsto v * x+(n-1) \mapsto v\end{array}\right\} & \text{if } i=1\\ \left\{\begin{array}{l}[\mathsf{a}_{i-1}]:\\ \exists v. x+(i-2) \mapsto v+1 * x+(i-1) \mapsto v * x+i \mapsto v\\ \rightsquigarrow x+(i-2) \mapsto v+1 * x+(i-1) \mapsto v+1 * x+i \mapsto v\end{array}\right\} & \text{otherwise}\end{cases}$$

Figure 5.11: The CoLoSL specification of an n-ary mutual exclusion ring.

5.3 N-ary Mutual Exclusion Ring

In this section we demonstrate how we can generalise the concepts introduced in §2 to generate an n-ary mutual exclusion device to manage concurrent accesses to a shared resource such as a database.

We generalise the ternary specification of Dijkstra's token ring described

in §2 to n places as presented in Fig. 5.11. The $\mathsf{ring}\,(x, i, n)$ predicate describes the members of the ring as n consecutive heap cells between address x and address $x + (n - 1)$, inclusively where the element at index i is the next to be incremented (hereafter, the leading element or component). That is, all components upto (but not including) index i have value v while those between i and $n - 1$ (end of the ring) have value $v - 1$ for some value v. The interference on the ring is the combined interference associated with each member of the ring $(I'(0) \cup \cdots \cup I'(n - 1))$.

While the $\mathsf{ring}\,(x, i, n)$ predicate provides a *global* account of all members of the ring, we also provide a *local* description of each of the components in the ring. The ith component of the n-ary ring at address x is captured by the $\mathsf{node}\,(x, i, n)$ predicate. As before, given a node at index i and its predecessor at index $(i - 1)\%n$ (where % denotes the remainder operation), at any given time either their values are equal, or the node at the lower index ($min((i - 1)\%n, i)$) is greater by one. The interference associated with the ith node consists of the updates allowed on the node itself $(I'(i))$ as well as the updates associated with its predecessor $(I''(i))$. The $\mathsf{turn}\,(x, i, n)$ predicate states that the leading component of the n-ary ring at address x resides at index i. That is, until the value of the member at index i is incremented no other component may be updated and as such, the thread associated with this element bears the sole right to manipulate the ring.

As we will shortly demonstrate, each of the n threads in the mutual exclusion ring maintains a local (subjective) view of the ring when interacting with it. That is, when a thread inspects the current state of the ring (in order to infer whether it is associated with the leading component) or when incrementing the value of its ring member, it does so locally by enquiring only the values of its own component and its immediate left neighbour. As such, the specification of these operations are also local and are concerned with the values of the node and its neighbour alone (see Fig. 5.16). On the other hand, the n-threaded concurrent database is equipped with the global specification of the mutual exclusion ring as to reflect the overall state of the ring and convey the identity of the leading thread (the thread next in line to access the database).

Fig. 5.12 presents a possible implementation for instantiating an n-place mutual exclusion ring along with an outline of the reasoning steps taken. Upon allocation of n consecutive cells at heap address x as the members of the ring, the shared state is extended by the ring components and their associated interference producing a *global* specification of the ring, similar to the initial ternary specification of §2. Using the Copy, Shift and Forget principles, we then obtain a *local* specification for each of the ring components

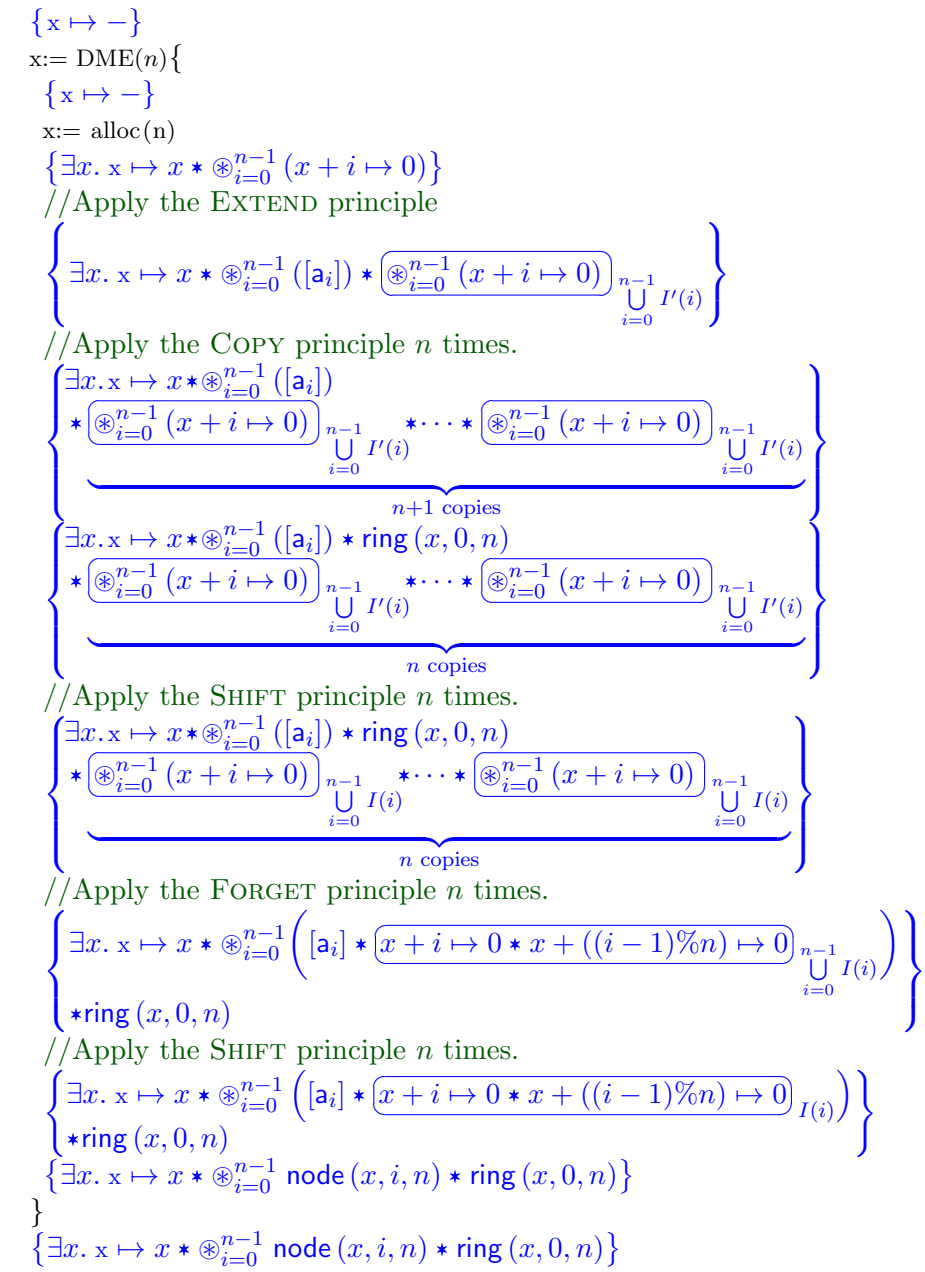

Figure 5.12: Instantiation of the n-ary mutual exclusion ring.

$$\mathsf{DB}\,(d,x,n) \stackrel{\text{def}}{=} \underbrace{\begin{array}{l}\exists i.\ \mathsf{ring}\,(x,i,n) \ast \\ ((\mathsf{data\text{-}base}(d) \ast d \Rrightarrow 0) \vee (d \Rrightarrow 1 \ast [\mathsf{a}_i]))\end{array}}_{I(d)}$$

$$\mathsf{data\text{-}base}(d) \stackrel{\text{def}}{=} d \mapsto -$$

$$I(d) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \left\{ [\mathsf{a}_i] : \left\{ \begin{array}{r} \mathsf{ring}\,(x,i,n) \ast \mathsf{data\text{-}base}(d) \ast d \Rrightarrow 0 \\ \rightsquigarrow \mathsf{ring}\,(x,i,n) \ast d \Rrightarrow 1 \ast [\mathsf{a}_i] \end{array} \right\} \right\}$$
$$\bigcup \left\{ true : \left\{ \begin{array}{r} \mathsf{ring}\,(x,i,n) \ast d \Rrightarrow 1 \ast [\mathsf{a}_i] \rightsquigarrow \\ \mathsf{ring}\,(x,i,n) \ast \mathsf{data\text{-}base}(d) \ast d \Rrightarrow 0 \end{array} \right\} \right\}$$

Figure 5.13: The CoLoSL specification of a shared database accessible by n threads synchronised through an n-ary mutual exclusion ring.

as captured by the final $\ast$-composed node predicates.

Fig. 5.16 presents two auxiliary methods for maintaining the n-ary ring at address x, namely isTurn(x,i,n) and next(x,i), along with their CoLoSL proof sketches. The isTurn(x,i,n) method can be used to ascertain whether the element at index i is the leading component (i.e. $\mathsf{turn}\,(x,i,n)$ holds); while next(x,i) is used to increment the value of the component at index i.

We proceed with the specification of a shared database concurrently accessed by n threads as presented in Fig. 5.13. The $\mathsf{DB}\,(d,x,n)$ predicate describes a database whose contents is captured by the $\mathsf{data\text{-}base}(d)$ predicate and concurrent access to it is synchronised by means of an n-ary mutual exclusion ring at address x as described above. An auxiliary capability $d \Rrightarrow -$ tracks the status of the database: $d \Rrightarrow 0$ denotes that the database is *idle* (not being accessed by any threads) and thus its contents ($\mathsf{data\text{-}base}(d)$) are in the shared state, visible to all threads. Conversely, $d \Rrightarrow 1$ indicates that the database is *busy* (under manipulation by a thread) and thus its contents have been claimed by the leading thread and are missing from the shared state. Moreover, the thread manipulating the database has temporarily turned its update capability ($[\mathsf{a}_i]$) to the shared state; this is used to track the identity of the leading thread.

The specification of the database itself ($\mathsf{data\text{-}base}(d)$) is orthogonal to the objective of this example and we thus represent it as a single heap cell at address d. In general, $\mathsf{data\text{-}base}(d)$ can be defined arbitrarily to capture the contents of a database at location d so long as it is disjoint from the mutual exclusion ring.

The interference associated with the database ($I(d)$), asserts that ith

thread (in the n-ary mutual exclusion ring at address x) associated with the leading component ($\mathsf{ring}\,(x, i, n)$) may change the status of the database from idle to busy and claim the contents of the database in return for its update capability [a_i]. That is, the leading thread at index i who possesses the sole right to manipulate the mutual exclusion ring, may remove the contents of the database from the shared state and update it locally. Conversely, once the leading thread has completed its operations on the database, it may return its contents to the shared state and reset its status from busy to idle while claiming its update capability.

We finalise this section by providing an example implementation of an n-threaded concurrent database and its proof sketch as shown in Fig. 5.14. At the beginning of the database program in Fig. 5.14 (concurrent-database(d,n)), the contents of the database is unshared and is held locally as captured by the $\mathsf{data\text{-}base}(d)$ predicate in the precondition. The program proceeds by first generating an n-ary mutual exclusion ring by calling the instantiation method (DME(n)). It then extends the shared state by the contents of the database and in doing so it makes the database a shared resource accessible by all n threads in the mutual exclusion ring at x. The program then continues by executing in parallel the programs associated with each of the n threads for concurrently manipulating the database (process([d],[x],i,[n]) for $i \in \{0, \cdots, n-1\}$).

The ith thread executing the (process(d,x,i,n)) program for manipulating the database, continuously *spins* until it becomes the leading thread. It then claims the database locally and proceeds with its intended modifications. Once complete, it returns the database to the shared state and advances the mutual exclusion ring by calling the next(x,i) method. It then resumes spinning and the same process repeats indefinitely.

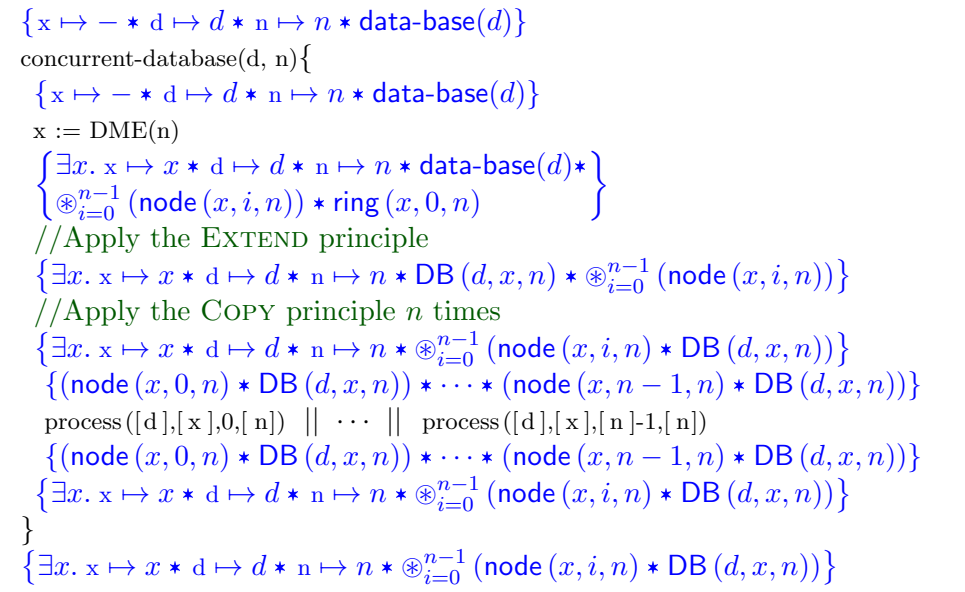

Figure 5.14: An n-threaded concurrent database where the implementation of the process method is given Fig. 5.15.

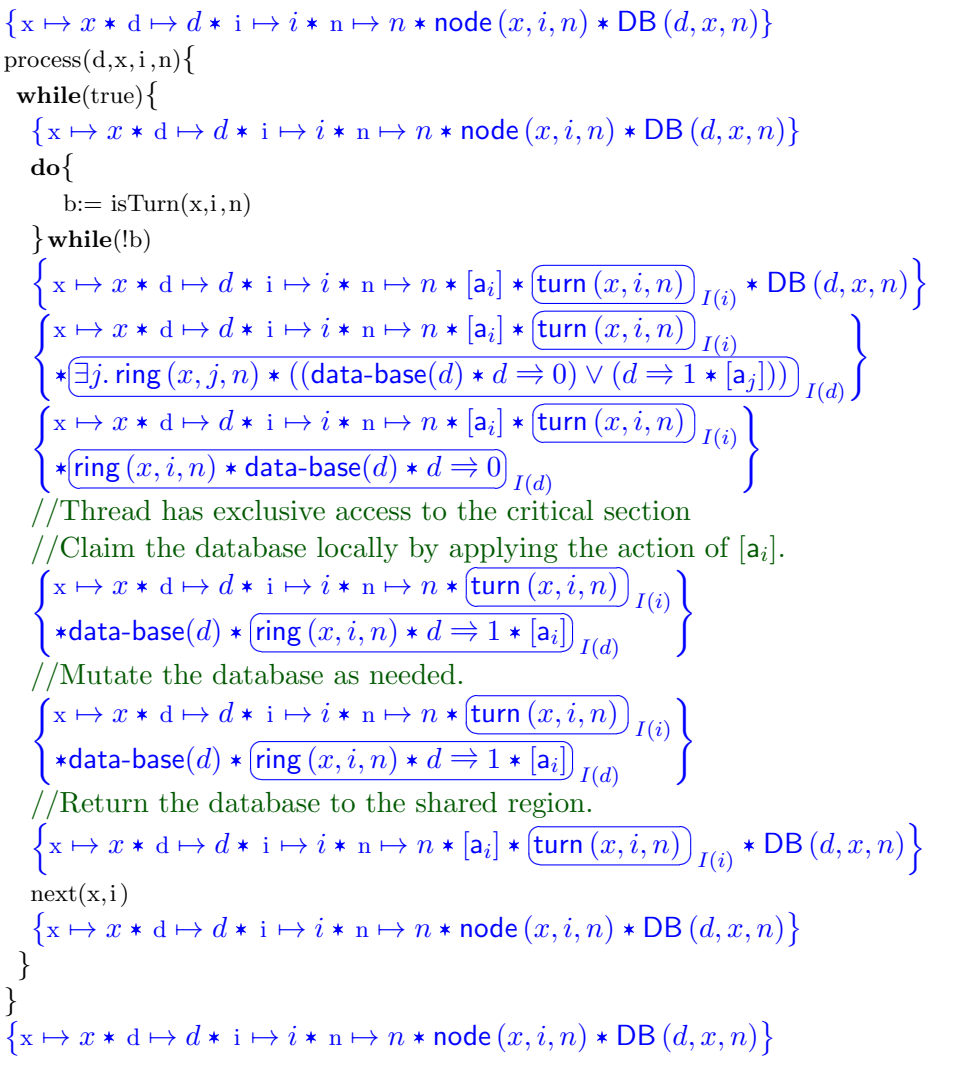

Figure 5.15: The code for each of the n threads of the concurrent database.

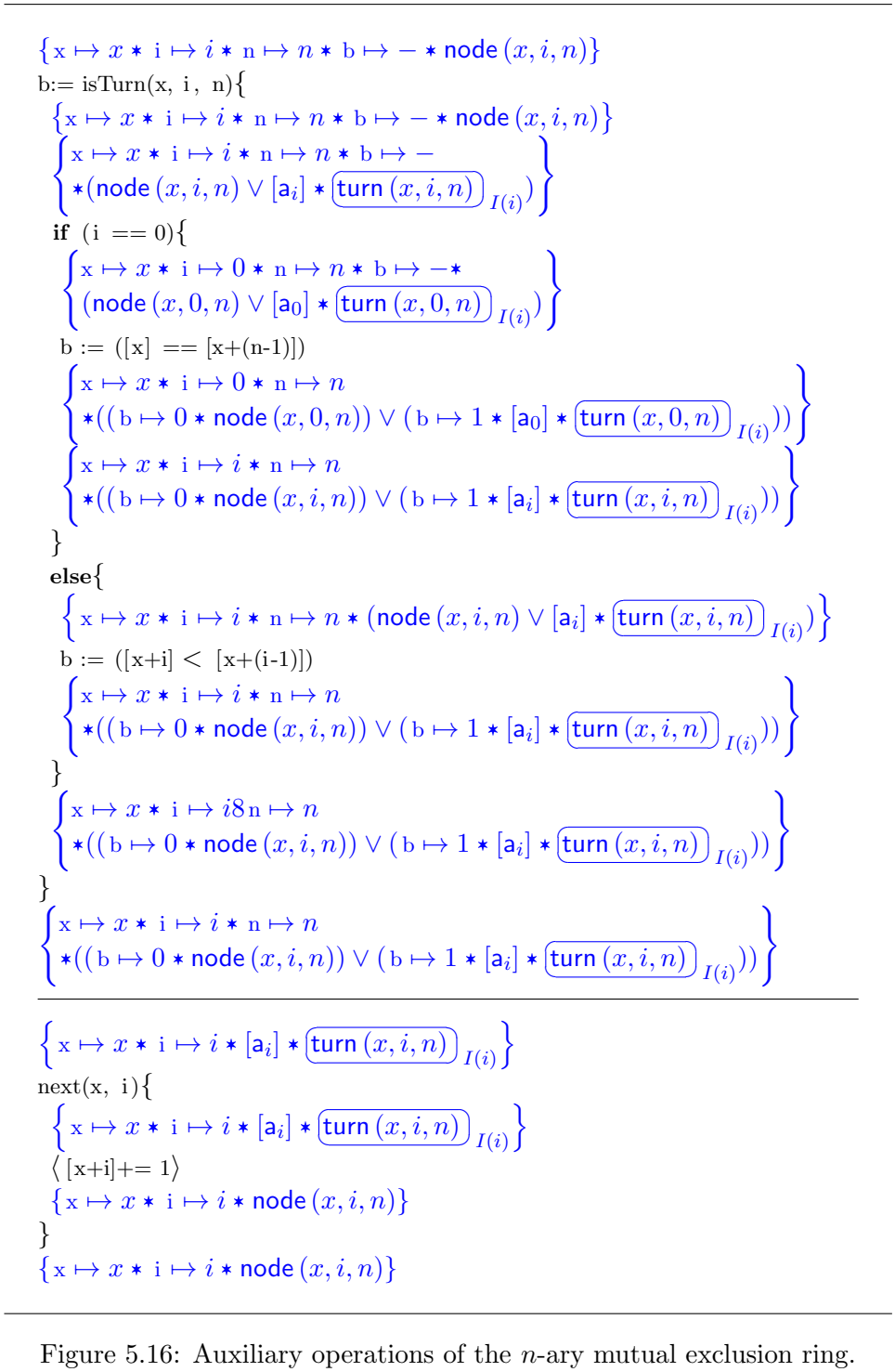

Figure 5.16: Auxiliary operations of the n-ary mutual exclusion ring.

Bibliography

[1] Richard Bornat, Cristiano Calcagno, and Hongseok Yang. Variables as resource in separation logic. *ENTCS*, 155:247–276, 2006.

[2] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *LICS*, pages 366–378, 2007.

[3] Pedro da Rocha Pinto, Thomas Dinsdale-Young, and Philippa Gardner. TaDA: A logic for time and data abstraction. In *ECOOP*, 2014.

[4] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[5] Thomas Dinsdale-Young, Lars Birkedal, Philippa Gardner, Matthew J. Parkinson, and Hongseok Yang. Views: compositional reasoning for concurrent programs. In *POPL*, pages 287–300, 2013.

[6] Thomas Dinsdale-Young, Mike Dodds, Philippa Gardner, Matthew J. Parkinson, and Viktor Vafeiadis. Concurrent abstract predicates. In Theo D'Hondt, editor, *ECOOP*, volume 6183 of *LNCS*, pages 504–528. Springer, 2010.

[7] Mike Dodds, Xinyu Feng, Matthew J. Parkinson, and Viktor Vafeiadis. Deny-guarantee reasoning. In Giuseppe Castagna, editor, *ESOP*, volume 5502 of *Lecture Notes in Computer Science*, pages 363–377. Springer, 2009.

[8] Xinyu Feng. Local rely-guarantee reasoning. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, pages 315–327. ACM, 2009.

[9] Philippa Gardner, Sergio Maffeis, and Gareth David Smith. Towards a program logic for JavaScript. In John Field and Michael Hicks, editors, *POPL*, pages 31–44. ACM, 2012.

[10] Aquinas Hobor and Jules Villard. The ramifications of sharing in data structures. In *POPL*, pages 523–536, 2013.

[11] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, pages 14–26, 2001.

[12] Cliff B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.

[13] Ruy Ley-Wild and Aleksandar Nanevski. Subjective auxiliary state for coarse-grained concurrency. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 561–574. ACM, 2013.

[14] Peter W. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007.

[15] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[16] John C. Reynolds. A short course on separation logic. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/fox-19/member/jcr/wwwaac2003/notes7.ps, 2003.

[17] Kasper Svendsen and Lars Birkedal. Impredicative concurrent abstract predicates. In Zhong Shao, editor, *ESOP*, volume 8410 of *Lecture Notes in Computer Science*, pages 149–168. Springer, 2014.

[18] Viktor Vafeiadis and Matthew J. Parkinson. A marriage of rely/guarantee and separation logic. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 256–271. Springer, 2007.

A. Auxiliary Lemmata

Lemma 6 (FORGET-Closure)**.** For all $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ and $s_1, s_2, r \in \mathsf{LState}$

$$\mathcal{I}{\downarrow}\,(s_1 \circ s_2, r, \mathcal{I}') \implies \mathcal{I}{\downarrow}\,(s_1, s_2 \circ r, \mathcal{I}')$$

Proof. Pick an arbitrary $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ and $s_1, s_2, r \in \mathsf{LState}$ such that

$$\mathcal{I}{\downarrow}\left(s_1 \circ s_2, r, \mathcal{I}'\right) \tag{A.1}$$

From the definition of $\downarrow$, it then suffices to show

$$\forall n \in \mathbb{N}.\, \mathcal{I}{\downarrow_n}\left(s_1, s_2 \circ r, \mathcal{I}'\right) \tag{A.2}$$

RTS. (A.2)
Rather than proving (A.2) directly, we first establish the following.

$$\forall n \in \mathbb{N}.\, \forall s_1, s_2, r \in \mathsf{LState}.\\ \mathcal{I}{\downarrow_n}\left(s_1 \circ s_2, r, \mathcal{I}'\right) \implies \mathcal{I}{\downarrow_n}\left(s_1, s_2 \circ r, \mathcal{I}'\right) \tag{A.3}$$

We can then despatch (A.2) from (A.1) and (A.3); since for an arbitrary $n \in \mathbb{N}$, from (A.1) and the definition of $\downarrow$ we have $\mathcal{I}{\downarrow_n}\,(s_1 \circ s_2, r, \mathcal{I}')$ and consequently from (A.3) we derive $\mathcal{I}{\downarrow_n}\,(s_1, s_2 \circ r, \mathcal{I}')$ as required.

RTS. (A.3)
We proceed by induction on the number of steps n.

Base case $n = 0$
Pick an arbitrary $s_1, s_2, r \in \mathsf{LState}$. We are then required to show $\mathcal{I}{\downarrow_0}$ $(s_1, s_2 \circ r, \mathcal{I}')$ which follows trivially from the definition of $\downarrow_0$.

Inductive Step Pick an arbitrary $n \in \mathbb{N}$ and $s_1, s_2, r \in \mathsf{LState}$ such that

$$\mathcal{I}{\downarrow_n}\left(s_1 \circ s_2, r, \mathcal{I}'\right) \tag{A.4}$$

$$\forall s_1, s_2, r \in \mathsf{LState}.$$
$$\mathcal{I}{\downarrow_{(n-1)}}\left(s_1 \circ s_2, r, \mathcal{I}'\right) \implies \mathcal{I}{\downarrow_{(n-1)}}\left(s_1, s_2 \circ r, \mathcal{I}'\right) \qquad \text{(I.H.)}$$

RTS.

$$\forall \kappa.\, \forall a \in \mathcal{I}'(\kappa).\, \mathit{potential}(a, s_1 \circ s_2 \circ r) \Rightarrow \mathit{reflected}(a, s_1 \circ s_2 \circ r, \mathcal{I}(\kappa)) \quad \text{(A.5)}$$
$$\forall \kappa.\, \forall a \in \mathcal{I}(\kappa).\, \mathit{potential}(a, s_1 \circ s_2 \circ r) \Rightarrow$$
$$(\mathit{reflected}(a, s_1 \circ s_2 \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s_1)) \quad \text{(A.6)}$$
$$\wedge\, \forall (s', r') \in a[s_1, s_2 \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\left(s', r', \mathcal{I}'\right)) \quad \text{(A.7)}$$

RTS. (A.5)
Pick an arbitrary κ and $a \in \mathcal{I}'(\kappa)$ such that $\mathit{potential}(a, s_1 \circ s_2 \circ r)$. From (A.4) and the definition of $\downarrow_n$ we then have $\mathit{reflected}(a, s_1 \circ s_2 \circ r, \mathcal{I}(\kappa))$ as required.

RTS. (A.6)
Pick an arbitrary κ and $a \in \mathcal{I}(\kappa)$ such that

$$\mathit{potential}(a, s_1 \circ s_2 \circ r)$$

Then from (A.4) we have:

$$\mathit{reflected}(a, s_1 \circ s_2 \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s_1 \circ s_2)$$

In the case of the first disjunct the desired result holds trivially. On the other hand in the case of the second disjunct from the definition of *visible* we have $\neg \mathit{visible}(a, s_1)$ as required.

RTS. (A.7)
Pick an arbitrary κ and $a = (p, q, c) \in \mathcal{I}(\kappa)$ such that

$$\mathit{potential}(a, s_1 \circ s_2 \circ r)$$

From (A.4) we then have:

$$\forall (s', r') \in a[s_1 \circ s_2, r].\, \mathcal{I}{\downarrow_{(n-1)}}\left(s', r', \mathcal{I}'\right) \qquad \text{(A.8)}$$

Pick an arbitrary (s', r') such that

$$(s', r') \in a[s_1, s_2 \circ r] \qquad \text{(A.9)}$$

Then from the definition of $a[s_1, s_2 \circ r]$ and by the cross-split property we know there exists $ps_1, ps_2, p_r, s_1', s_2', r''' \in \mathsf{LState}$ such that :

$$\begin{array}{l} p = ps_1 \circ ps_2 \circ p_r \wedge s_1 = ps_1 \circ s_1' \wedge s_2 = ps_2 \circ s_2' \wedge r = p_r \circ r'' \wedge \\ \begin{pmatrix} (ps_1 > \mathbf{0_L} \wedge s' = q \circ s_1' \wedge r' = s_2' \circ r'') \\ \vee (ps_1 = \mathbf{0_L} s' = s_1 \wedge r' = q \circ s_2' \circ r'') \end{pmatrix} \end{array} \tag{A.10}$$

and consequently from the definition of $a[s_1 \circ s_2, r]$

$$\begin{array}{c} p = ps_1 \circ ps_2 \circ p_r \wedge s_1 = ps_1 \circ s_1' \wedge s_2 = ps_2 \circ s_2' \wedge r = p_r \circ r'' \wedge \\ \left(\begin{array}{l} (s' = q \circ s_1' \wedge r' = s_2' \circ r'' \wedge (q \circ s_1' \circ s_2', r'') \in a[s_1 \circ s_2, r]) \\ \vee \left(\begin{array}{l} ps_1 = \mathbf{0_L} \wedge s' = s_1 \wedge r' = q \circ s_2' \circ r'' \wedge \\ \left(\begin{array}{l} (ps_2 = \mathbf{0_L} \wedge (s_1' \circ s_2', q \circ r'') \in a[s_1 \circ s_2, r]) \\ \vee (ps_2 > \mathbf{0_L} \wedge (s_1' \circ q \circ s_2', r'') \in a[s_1 \circ s_2, r]) \end{array} \right) \end{array} \right) \end{array} \right) \end{array}$$

That is,

$$\exists s''.\, (s' \circ s'', r' - s'') \in a[s_1 \circ s_2, r] \tag{A.11}$$

From (A.8) and (A.11) we then have

$$\exists s''.\, \mathcal{I}{\downarrow_{(n-1)}}\ \left(s' \circ s'', r' - s'', \mathcal{I}'\right)$$

Finally from (I.H.) we have

$$\exists s''.\, \mathcal{I}{\downarrow_{(n-1)}}\ \left((s', (r' - s'') \circ s'', \mathcal{I}'\right)$$

That is,

$$\mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}'\right)$$

as required. □

Lemma 7 (MERGE-Closure)**.** For all $\mathcal{I}, \mathcal{I}_1, \mathcal{I}_2 \in \mathsf{AMod}$ and $s_p, s_c, s_q, r \in \mathsf{LState}$,

$$\mathcal{I}{\downarrow}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \wedge \mathcal{I}{\downarrow}\,(s_q \circ s_c, s_p \circ r, \mathcal{I}_2) \implies \mathcal{I}{\downarrow}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2)$$

Proof. Pick an arbitrary $\mathcal{I}, \mathcal{I}_1, \mathcal{I}_2 \in \mathsf{AMod}$ and $s_p, s_c, s_q, r \in \mathsf{LState}$ such that

$$\mathcal{I}{\downarrow}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \tag{A.12}$$

$$\mathcal{I}{\downarrow}\,(s_q \circ s_c, s_p \circ r, \mathcal{I}_2) \tag{A.13}$$

From the definition of $\downarrow$, it then suffices to show

$$\forall n \in \mathbb{N}.\, \mathcal{I}{\downarrow_n}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2) \tag{A.14}$$

RTS. (A.14)
Rather than proving (A.14) directly, we first establish the following.

$$\begin{aligned}&\forall n \in \mathbb{N}.\, \forall s_p, s_c, s_q, r \in \mathsf{LState}.\\ &\quad \mathcal{I}{\downarrow_n}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \wedge \mathcal{I}{\downarrow_n}\,(s_c \circ s_q, s_p \circ r, \mathcal{I}_2)\\ &\qquad \implies \mathcal{I}{\downarrow_n}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2)\end{aligned} \tag{A.15}$$

We can then despatch (A.14) from (A.12), (A.13) and (A.15); since for an arbitrary $n \in \mathbb{N}$, from (A.12) and the definition of $\downarrow$ we have $\mathcal{I}{\downarrow_n}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \wedge \mathcal{I}{\downarrow_n}\,(s_c \circ s_q, s_p \circ r, \mathcal{I}_2)$ and consequently from (A.15) we derive $\mathcal{I}{\downarrow_n}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2)$ as required.

RTS. (A.15)
We proceed by induction on the number of steps n.

Base case $n = 0$
Pick an arbitrary $s_1, s_2, r \in \mathsf{LState}$. We are then required to show $\mathcal{I}{\downarrow_0}\,(s_1, s_2 \circ r, \mathcal{I}')$ which follows trivially from the definition of $\downarrow_0$.

Inductive Step Pick an arbitrary $s_p, s_q, s_c, r \in \mathsf{LState}$ and $n \in \mathbb{N}$, such that

$$\mathcal{I}{\downarrow_n}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \tag{A.16}$$

$$\mathcal{I}{\downarrow_n}\,(s_q \circ s_c, s_p \circ r, \mathcal{I}_2) \tag{A.17}$$

$$\begin{aligned}&\forall s_p, s_q, s_c, r \in \mathsf{LState}.\\ &\quad \mathcal{I}{\downarrow_{(n-1)}}\,(s_p \circ s_c, s_q \circ r, \mathcal{I}_1) \;\wedge\; \mathcal{I}{\downarrow_{(n-1)}}\,(s_q \circ s_c, s_p \circ r, \mathcal{I}_2)\\ &\qquad \implies \mathcal{I}{\downarrow_{(n-1)}}\,(s_p \circ s_c \circ s_q, r, \mathcal{I}_1 \cup \mathcal{I}_2)\end{aligned} \tag{I.H.}$$

RTS.

$$\forall \kappa.\, \forall a \in (\mathcal{I}_1 \cup \mathcal{I}_2)(\kappa).\, \mathit{potential}(a, s_p \circ s_c \circ s_q \circ r) \Rightarrow \\ \mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, \mathcal{I}(\kappa)) \tag{A.18}$$

$$\forall \kappa.\, \forall a \in \mathcal{I}(\kappa).\, \mathit{potential}(a, s_p \circ s_c \circ s_q \circ r) \Rightarrow \\ (\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, (\mathcal{I}_1 \cup \mathcal{I}_2)\,(\kappa)) \vee \neg \mathit{visible}(a, s_p \circ s_c \circ s_q)\ ) \wedge \\ \forall (s', r') \in a[s_p \circ s_c \circ s_q, r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_1 \cup \mathcal{I}_2)) \tag{A.19}$$

RTS. (A.18)
Pick an arbitrary κ and $a \in (\mathcal{I}_1 \cup \mathcal{I}_2)(\kappa)$. From the definition of $\mathcal{I}_1 \cup \mathcal{I}_2$ we know either $a \in \mathcal{I}_1(\kappa)$ or $a \in \mathcal{I}_2(\kappa)$. In the former case from (A.16) and the definition of $\downarrow_n$ we have $\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, \mathcal{I}(\kappa))$ as needed. In the latter case from (A.17) and the definition of $\downarrow_n$ we have $\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, \mathcal{I}(\kappa))$ as required.

RTS. (A.19)
Pick an arbitrary κ and $a = (p, q, c) \in \mathcal{I}(\kappa)$ such that

$$\mathit{potential}(a, s_p \circ s_c \circ s_q \circ r) \tag{A.20}$$

From (A.16) and (A.20) we have:

$$(\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, \mathcal{I}_1(\kappa)) \vee \neg \mathit{visible}(a, s_p \circ s_c)) \wedge \\ \forall (s', r') \in a[s_p \circ s_c, s_q \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_1)$$

and consequently from the definition of $\mathcal{I}_1 \cup \mathcal{I}_2$ we have:

$$(\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, (\mathcal{I}_1 \cup \mathcal{I}_2)\,(\kappa)) \vee \neg \mathit{visible}(a, s_p \circ s_c)) \wedge \\ \forall (s', r') \in a[s_p \circ s_c, s_q \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_1) \tag{A.21}$$

Similarly, from (A.17) and (A.20) we have:

$$(\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, (\mathcal{I}_1 \cup \mathcal{I}_2)\,(\kappa)) \vee \neg \mathit{visible}(a, s_c \circ s_q)) \wedge \\ \forall (s', r') \in a[s_c \circ s_q, s_p \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_2) \tag{A.22}$$

From (A.21), (A.22) and the definition of *visible* we have:

$$(\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, (\mathcal{I}_1 \cup \mathcal{I}_2)\,(\kappa)) \vee \neg \mathit{visible}(a, s_p \circ s_c \circ s_q)) \wedge \\ \forall (s', r') \in a[s_p \circ s_c, s_q \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_1) \wedge \\ \forall (s', r') \in a[s_c \circ s_q, s_p \circ r].\, \mathcal{I}{\downarrow_{(n-1)}}\,(s', r', \mathcal{I}_2) \tag{A.23}$$

Pick an arbitrary $s', r' \in \mathsf{LState}$ such that

$$(s', r') \in a[s_p \circ s_c \circ s_q, r] \tag{A.24}$$

Then from the definition of $a[s_p \circ s_c \circ s_q]$ and by the cross-split property we know there exists $p_p, p_c, p_q, s'_p, s'_c, s'_q \in \mathsf{LState}$ such that :

$$\begin{array}{l} s' = s_p \circ s_c \circ s_q \vee \\ \left(\begin{array}{l} (p_p > \mathbf{0}_\mathsf{L} \vee p_c > \mathbf{0}_\mathsf{L} \vee p_q > \mathbf{0}_\mathsf{L}) \wedge s' = q \circ s'_p \circ s'_c \circ s'_q \\ \wedge p = p_p \circ p_c \circ p_q \circ p_r \\ \wedge s_p = p_p \circ s'_p \wedge s_c = p_c \circ s'_c \wedge s_q = p_q \circ s'_q \wedge r = p_r \circ r' \end{array} \right) \end{array} \tag{A.25}$$

and consequently from the definitions of $a[s_p \circ s_c, s_q \circ r]$ and $a[s_c \circ s_q, s_p \circ r]$ we have:

$$\begin{array}{l} \left(\begin{array}{l} s' = s_p \circ s_c \circ s_q \wedge \\ (s_p \circ s_c, s_q \circ r') \in a[s_p \circ s_c, s_q \circ r] \wedge (s_c \circ s_q, s_p \circ r') \in a[s_c \circ s_q, s_p \circ r] \end{array} \right) \\ \vee \left(\begin{array}{l} s' = q \circ s'_p \circ s'_c \circ s'_q \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} ((p_c > \mathbf{0}_\mathsf{L} \vee (p_c = \mathbf{0}_\mathsf{L} \wedge p_p > \mathbf{0}_\mathsf{L} \wedge p_q > \mathbf{0}_\mathsf{L}))) \wedge \\ (q \circ s'_p \circ s'_c, s'_q \circ r') \in a[s_p \circ s_c, s_q \circ r] \wedge \\ (q \circ s'_c \circ s'_q, s'_p \circ r') \in a[s_c \circ s_q, s_p \circ r] \end{array} \right) \\ \vee \left(\begin{array}{l} p_p > \mathbf{0}_\mathsf{L} \wedge p_c = p_q = \mathbf{0}_\mathsf{L} \wedge \\ (q \circ s'_p \circ s'_c, s'_q \circ r') \in a[s_p \circ s_c, s_q \circ r] \wedge \\ (s'_c \circ s'_q, q \circ s'_p \circ r') \in a[s_c \circ s_q, s_p \circ r] \end{array} \right) \\ \vee \left(\begin{array}{l} p_q > \mathbf{0}_\mathsf{L} \wedge p_c = p_p = \mathbf{0}_\mathsf{L} \wedge \\ (s'_p \circ s'_c, q \circ s'_q \circ r') \in a[s_p \circ s_c, s_q \circ r] \wedge \\ (q \circ s'_c \circ s'_q, s'_p \circ r') \in a[s_c \circ s_q, s_p \circ r] \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \tag{A.26}$$

From (A.26), (A.16), (A.17) and (A.20) we have:

$$\begin{array}{l} \left(\begin{array}{l} s' = s_p \circ s_c \circ s_q \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \ (s_p \circ s_c, s_q \circ r', \mathcal{I}_1) \wedge \mathcal{I}{\downarrow_{(n-1)}} \ (s_c \circ s_q, s_p \circ r', \mathcal{I}_2) \end{array} \right) \\ \vee \left(\begin{array}{l} s' = q \circ s'_p \circ s'_c \circ s'_q \wedge \\ \left(\begin{array}{l} \left(\begin{array}{l} ((p_c > \mathbf{0}_\mathsf{L} \vee (p_c = \mathbf{0}_\mathsf{L} \wedge p_p > \mathbf{0}_\mathsf{L} \wedge p_q > \mathbf{0}_\mathsf{L}))) \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(q \circ s'_p \circ s'_c, s'_q \circ r', \mathcal{I}_1\right) \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(q \circ s'_c \circ s'_q, s'_p \circ r', \mathcal{I}_2\right) \end{array} \right) \\ \vee \left(\begin{array}{l} p_p > \mathbf{0}_\mathsf{L} \wedge p_c = p_q = \mathbf{0}_\mathsf{L} \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(q \circ s'_p \circ s'_c, s'_q \circ r', \mathcal{I}_1\right) \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(s'_c \circ s'_q, q \circ s'_p \circ r', \mathcal{I}_2\right) \end{array} \right) \\ \vee \left(\begin{array}{l} p_q > \mathbf{0}_\mathsf{L} \wedge p_c = p_p = \mathbf{0}_\mathsf{L} \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(s'_p \circ s'_c, q \circ s'_q \circ r', \mathcal{I}_1\right) \wedge \\ \mathcal{I}{\downarrow_{(n-1)}} \left(q \circ s'_c \circ s'_q, s'_p \circ r', \mathcal{I}_2\right) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \tag{A.27}$$

and thus from (A.27) and (I.H.)

$$\mathcal{I}{\downarrow_{(n-1)}}\left(s', r', \mathcal{I}_1 \cup \mathcal{I}_2\right) \tag{A.28}$$

Finally, from (A.23), (A.24) and (A.28) we have:

$$(\mathit{reflected}(a, s_p \circ s_c \circ s_q \circ r, (\mathcal{I}_1 \cup \mathcal{I}_2)(\kappa)) \vee \neg \mathit{visible}(a, s_p \circ s_c \circ s_q)) \wedge \\ \forall (s', r') \in a[s_p \circ s_c \circ s_q, r].\, \mathcal{I}{\downarrow_{(n-1)}}\, (s', r', \mathcal{I}_1 \cup \mathcal{I}_2)$$

as required.

□

Lemma 8 (SHIFT-Fence)**.** For all $\mathcal{I}_1, \mathcal{I}_2 \in \mathsf{AMod}$, $s, s', r \in \mathsf{LState}$ and $a \in rg(\mathcal{I}_1)$:

$$\mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \wedge \big(s' \in a(s) \vee (s', -) \in a[s, r]\big) \implies \mathcal{I}_1 \sqsubseteq^{\{s'\}} \mathcal{I}_2$$

Proof. Pick an arbitrary $\mathcal{I}_1, \mathcal{I}_2 \in \mathsf{AMod}$, $s, s', r \in \mathsf{LState}$ and $a \in rg(\mathcal{I}_1)$ such that:

$$\mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \tag{A.29}$$

RTS. $\mathcal{I}_1 \sqsubseteq^{\{s'\}} \mathcal{I}_2$.
There are two cases to consider:
Case 1. $s' \in a(s)$
From the definition of $\sqsubseteq^{\{s\}}$ and (A.29) we know there exists a fence $\mathcal{F}$ such that:

$$s \in \mathcal{F} \tag{A.30}$$

$$\mathcal{F} \rhd \mathcal{I}_1 \tag{A.31}$$

$$\forall l \in \mathcal{F}.\, \forall \kappa.\, \forall a \in \mathcal{I}_2(\kappa).\, \mathit{reflected}(a, l, \mathcal{I}_1(\kappa)) \wedge \forall a \in \mathcal{I}_1(\kappa).\, a(l) \text{ is defined } \wedge \mathit{visible}(a, l) \implies \mathit{reflected}(a, l, \mathcal{I}_2(\kappa)) \tag{A.32}$$

By definition of $\rhd$ and from (A.30)-(A.31) and assumption of case 1. we have:

$$s' \in \mathcal{F} \tag{A.33}$$

Finally by definition of $\sqsubseteq^{\{s'\}}$ and (A.31)-(A.33) we have

$$\mathcal{I}_1 \sqsubseteq^{\{s'\}} \mathcal{I}_2$$

as required.

Case 2. $(s', -) \in a[s, r]$
From the definitions of $a[s, r]$ and $a(s)$ and from the assumption of the case we know $s' \in a(s)$. The rest of the proof is identical to that of case 1. □

Lemma 9 (SHIFT-Closure)**.** For all $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I} \in \mathsf{AMod}$ and $s, r \in \mathsf{LState}$,

$$\mathcal{I}{\downarrow}\,(s, r, \mathcal{I}_1) \wedge \mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \implies \mathcal{I}{\downarrow}\,(s, r, \mathcal{I}_2)$$

Proof. Pick an arbitrary $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I} \in \mathsf{AMod}$ and $s, r \in \mathsf{LState}$ such that

$$\mathcal{I}{\downarrow}\ (s, r, \mathcal{I}_1) \tag{A.34}$$

$$\mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \tag{A.35}$$

From the definition of $\downarrow$, it then suffices to show

$$\forall n \in \mathbb{N}.\, \mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_2) \tag{A.36}$$

RTS. (A.36)
Rather than proving (A.36) directly, we first establish the following.

$$\begin{aligned} &\forall n \in \mathbb{N}.\, \forall s, r \in \mathsf{LState}. \\ &\quad \mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_1) \wedge \mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \implies \mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_2) \end{aligned} \tag{A.37}$$

We can then despatch (A.36) from (A.34), (A.35) and (A.37); since for an arbitrary $n \in \mathbb{N}$, from (A.34) and the definition of $\downarrow$ we have $\mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_1)$ and consequently from (A.35) and (A.37) we derive $\mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_2)$ as required.

RTS. (A.37)
We proceed by induction on the number of steps n.

Base case $n = 0$
Pick an arbitrary $s, r \in \mathsf{LState}$. We are then required to show $\mathcal{I}{\downarrow_0}\ (s, r, \mathcal{I}_2)$ which follows trivially from the definition of $\downarrow_0$.

Inductive Case
Pick an arbitrary $s, r \in \mathsf{LState}$ such that:

$$\mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_1) \tag{A.38}$$

$$\mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \tag{A.39}$$

$$\begin{aligned} &\forall s, r \in \mathsf{LState}. \\ &\quad \mathcal{I}{\downarrow_{(n-1)}}\ (s, r, \mathcal{I}_1) \wedge \mathcal{I}_1 \sqsubseteq^{\{s\}} \mathcal{I}_2 \implies \mathcal{I}{\downarrow_{(n-1)}}\ (s, r, \mathcal{I}_2) \end{aligned} \tag{I.H}$$

RTS.

$$\forall \kappa.\, \forall a \in \mathcal{I}_2(\kappa).\, \mathit{potential}(a, s \circ r) \Rightarrow \mathit{reflected}(a, s \circ r, \mathcal{I}(\kappa)) \tag{A.40}$$

$$\begin{aligned} &\forall \kappa.\, \forall a \in \mathcal{I}(\kappa).\, \mathit{potential}(a, s \circ r) \Rightarrow \\ &\quad (\mathit{reflected}(a, s \circ r, \mathcal{I}_2(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge \\ &\quad \forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}_2) \end{aligned} \tag{A.41}$$

RTS. (A.40)
Pick an arbitrary κ and $a = (p, q, c) \in \mathcal{I}_2(\kappa)$ such that:

$$\mathit{potential}(a, s \circ r)$$

Pick an arbitrary $l \in \mathsf{LState}$ such that

$$p \circ c \leq s \circ r \circ l \tag{A.42}$$

From (A.39 and the definition of *reflected* we then know there exists a'', c'' such that

$$a'' = (p, q, c'') \in \mathcal{I}_1(\kappa) \wedge p \circ c'' \leq s \circ r \circ l$$

Consequently from (A.38) and the definition of *reflected* we know there exists a', c' such that

$$a' = (p, q, c') \in \mathcal{I}(\kappa) \wedge p \circ c' \leq s \circ r \circ l \tag{A.43}$$

Thus from (A.42),(A.43) and the definition of *reflected* we have:

$$\mathit{reflected}(a, s \circ r, \mathcal{I}(\kappa))$$

as required.

RTS. (A.41)
Pick an arbitrary κ and $a = (p, q, c) \in \mathcal{I}(\kappa)$ such that:

$$\mathit{potential}(a, s \circ r) \tag{A.44}$$

From (A.38) and (A.44) we have:

$$\begin{array}{l}(\mathit{reflected}(a, s \circ r, \mathcal{I}_1(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge \\ \forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}_1)\end{array} \tag{A.45}$$

Pick an arbitrary (s', r') such that

$$(s', r') \in a[s, r] \tag{A.46}$$

Then from (A.45) we have:

$$\mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_1\right) \tag{A.47}$$

On the other hand, from (A.46) and Lemma 8 we have:

$$\mathcal{I}_1 \sqsubseteq^{\{s'\}} \mathcal{I}_2 \tag{A.48}$$

Consequently, from (A.47), (A.48) and (I.H) we have:

$$\mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_2\right)$$

and thus from (A.46) we have

$$\forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_2\right) \tag{A.49}$$

Since either $\mathit{visible}(a, s)$ or $\neg \mathit{visible}(a, s)$, there are two cases to consider:

Case 1. $\neg \mathit{visible}(a, s)$
From the assumption of case 1 and (A.49) we then have

$$\begin{aligned} &\neg \mathit{visible}(a, s) \wedge \\ &\forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_2\right) \end{aligned}$$

as required.

Case 2. $\mathit{visible}(a, s)$
From (A.45) and the assumption of case 2 we then have

$$\mathit{reflected}(a, s \circ r, \mathcal{I}_1(\kappa)) \tag{A.50}$$

Pick an arbitrary $l \in \mathsf{LState}$ such that:

$$p \circ c \leq s \circ r \circ l \tag{A.51}$$

Then from (A.50) and the definition of *reflected* we have:

$$\exists a', c'.\, a' = (p, q, c') \wedge a' \in \mathcal{I}_1(\kappa) \wedge p \circ c' \leq s \circ r \circ l \tag{A.52}$$

From (A.44) and by definition of *potential* we know $a[s \circ r]$ is defined; from (A.52), and the definition of $a'[s \circ r]$ we know that $a'[s \circ r]$ is also defined. Consequently, from the definition of $a'(s)$, we know $a'(s)$ is also defined. Thus, from the definition of *visible*, (A.52) and the assumption of case 1.2 we have

$$\mathit{visible}(a', s) \tag{A.53}$$

Thus from (A.39), (A.52), (A.53) and from the definition of $\sqsubseteq^{\{s\}}$ we have

$$\exists a'', c''.\, a'' = (p, q, c'') \wedge a'' \in \mathcal{I}_2(\kappa) \wedge p \circ c'' \leq s \circ r \circ l \tag{A.54}$$

Finally, from (A.51), (A.52), (A.54) and by definition of *reflected* we have:

$$\mathit{reflected}(a, s \circ r, \mathcal{I}_2(\kappa)) \tag{A.55}$$

From (A.49) and (A.55) we have

$$\begin{array}{l} \mathit{reflected}(a, s \circ r, \mathcal{I}_2(\kappa)) \wedge \\ \forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_2\right) \end{array}$$

as required.

□

Lemma 10 (EXTEND-Closure-1)**.** For all $\mathcal{I}, \mathcal{I}_e, \mathcal{I}_0 \in \mathsf{AMod}$ such that $\forall \kappa \in \mathsf{dom}\,(\mathcal{I}_0)\,.\,\mathcal{I}_0(\kappa) = \emptyset$; and for all $g, s_e \in \mathsf{LState}$ and $\mathcal{F}, \mathcal{F}_e \in \mathcal{P}\,(\mathsf{LState})$,

$$g \in \mathcal{F} \wedge \mathcal{F} \blacktriangleright \mathcal{I} \wedge s_e \in \mathcal{F}_e \wedge \mathcal{F}_e \blacktriangleright \mathcal{I}_e \cup \mathcal{I}_0 \implies \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow}\,(s_e, g, \mathcal{I}_e)$$

Proof. Pick an arbitrary $\mathcal{I}, \mathcal{I}_e, \mathcal{I}_0 \in \mathsf{AMod}$, $g, s_e \in \mathsf{LState}$ and $\mathcal{F}, \mathcal{F}_e \in \mathcal{P}\,(\mathsf{LState})$ such that

$$\forall \kappa \in \mathsf{dom}\,(\mathcal{I}_0)\,.\,\mathcal{I}_0(\kappa) = \emptyset \tag{A.56}$$
$$g \in \mathcal{F} \wedge s_e \in \mathcal{F}_e \tag{A.57}$$
$$\mathcal{F} \blacktriangleright \mathcal{I} \wedge \mathcal{F}_e \blacktriangleright \mathcal{I}_e \cup \mathcal{I}_0 \tag{A.58}$$

From the definition of $\downarrow$, it then suffices to show

$$\forall n \in \mathbb{N}.\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_n}\,(s_e, g, \mathcal{I}_e) \tag{A.59}$$

RTS. (A.59)
Rather than proving (A.59) directly, we first establish the following.

$$\begin{aligned}&\forall n \in \mathbb{N}.\, \forall g, s_e \in \mathsf{LState}.\\ &\quad g \in \mathcal{F} \wedge s_e \in \mathcal{F}_e \implies \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_n}\,(s_e, g, \mathcal{I}_e)\end{aligned} \tag{A.60}$$

We can then despatch (A.59) from (A.57) and (A.60); since for an arbitrary $n \in \mathbb{N}$, from (A.57) and (A.60) we have $\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_n}\,(s_e, g, \mathcal{I}_e)$ as required.

RTS. (A.60)
We proceed by induction on the number of steps n.

Base case $n = 0$
Pick an arbitrary $g, s_e \in \mathsf{LState}$. We are then required to show $\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_0}\,(s_e, g, \mathcal{I}_e)$ which follows trivially from the definition of $\downarrow_0$.

Inductive case
Pick an arbitrary $n \in \mathbb{N}$ and $g, s_e \in \mathsf{LState}$ such that

$$g \in \mathcal{F} \tag{A.61}$$
$$s_e \in \mathcal{F}_e \tag{A.62}$$
$$\forall g', s_e'.\, g' \in \mathcal{F} \wedge s_e' \in \mathcal{F}_e \implies \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}}\,\left(s_e', g', \mathcal{I}_e\right) \tag{I.H}$$

RTS.

$$\forall \kappa.\, \forall a \in \mathcal{I}_e(\kappa).\, \mathit{potential}(a, s_e \circ g) \implies \mathit{reflected}(a, s_e \circ g, (\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0)(\kappa)) \tag{A.63}$$

$$\forall\kappa.\,\forall a \in (\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0)(\kappa).\, \mathit{potential}(a, s_e \circ g) \implies \\ (\mathit{reflected}(a, s_e \circ g, \mathcal{I}_e(\kappa)) \vee \neg \mathit{visible}(a, s_e)\ ) \wedge \\ \forall (s', r') \in a[s_e, g].\, \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}_e) \tag{A.64}$$

RTS. A.63
Pick an arbitrary κ, $a \in \mathcal{I}_e(\kappa)$ such that $\mathit{potential}(a, s_e \circ g)$. From the definitions of $\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0$ and *reflected* we then trivially have $\mathit{reflected}(a, s_e \circ g, (\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0)(\kappa))$ as required.

RTS. A.64
Pick an arbitrary κ, $a = (p, q, c) \in (\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0)(\kappa)$ and (s', r') such that

$$\mathit{potential}(a, s_e \circ g) \tag{A.65}$$
$$(s', r') \in a[s_e, g] \tag{A.66}$$

Since from (A.56) we know $\mathcal{I}_0(\kappa) = \emptyset$ we know $a \in \mathcal{I}_e(\kappa) \vee a \in \mathcal{I}(\kappa)$, and thus there are two cases to consider.

Case 1. $a \in \mathcal{I}(\kappa)$
From (A.65) and the definition of *potential* we have $s_e \circ g \sqcap p \circ c \neq \emptyset$ and consequently, $g \sqcap p \circ c \neq \emptyset$, from (A.61) we then have:

$$p \leq g \wedge p \perp s_e \tag{A.67}$$

From (A.65), (A.67) and the definitions of *potential* we have:

$$\mathit{potential}(a, g) \tag{A.68}$$

From (A.67) and the definition of *visible* we have:

$$\neg \mathit{visible}(a, s_e) \tag{A.69}$$

On the other hand, from (A.66), (A.67) and the definitions of $a[s_e, g]$, $a[g]$ and $\perp$, we know:

$$s' = s_e \tag{A.70}$$
$$a[g] = r' \tag{A.71}$$

Consequently, from (A.58), (A.61), (A.68), (A.71) and the definition of $\blacktriangleright$ we have:

$$r' \in \mathcal{F} \tag{A.72}$$

Finally, from (A.62), (A.72), (I.H) we have:

$$\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}} \left(s_e, r', \mathcal{I}_e\right) \tag{A.73}$$

and consequently from (A.69) and (A.66)-(A.73) we have

$$\begin{array}{l} \neg \mathit{visible}(a, s_e) \wedge \\ \forall (s', r') \in a[s_e, g].\, \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}} \left(s_e, r', \mathcal{I}_e\right) \end{array}$$

as required.

Case 2. $a \in \mathcal{I}_e(\kappa)$
From the assumption of the case and the definition of *reflected* we trivially have

$$\mathit{reflected}(a, s_e \circ g, \mathcal{I}_e(\kappa) \tag{A.74}$$

Since from (A.65) and the definition of *potential* we have $s_e \circ g \sqcap p \circ c \neq \emptyset$ and consequently, $s_e \sqcap p \circ c \neq \emptyset$, from (A.62) we have:

$$p \leq s_e \wedge p \perp g \tag{A.75}$$

From (A.65), (A.75) and the definition of *potential* we have:

$$\mathit{potential}(a, s_e) \tag{A.76}$$

On the other hand, from (A.66), (A.75) and the definitions of $a[s_e, g]$ and $\perp$, we have:

$$r' = g \tag{A.77}$$

$$a[s_e] = s' \tag{A.78}$$

Consequently, from (A.58), (A.62), (A.76), (A.78) and the definition of $\blacktriangleright$ we have:

$$s' \in \mathcal{F}_e \tag{A.79}$$

Finally, from (A.61), (A.79) and (I.H) we have:

$$\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}} \left(s', g, \mathcal{I}_e\right)$$

and consequently from (A.77)

$$\mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_e\right) \tag{A.80}$$

Thus from (A.66) and (A.74)-(A.80) we have:

$$\begin{aligned}&\mathit{reflected}(a, s_e \circ g, \mathcal{I}_e(\kappa)) \wedge \\ &\forall (s', r') \in a[s_e, g].\, \mathcal{I} \cup \mathcal{I}_e \cup \mathcal{I}_0 \downarrow_{(n-1)} \left(s', r', \mathcal{I}_e\right)\end{aligned}$$

as required.

□

Lemma 11 (EXTEND-closure-2)**.** For all $\mathcal{I}_0, \mathcal{I}, \mathcal{I}_e \in \mathsf{AMod}$, $s, g, s_e \in \mathsf{LState}$ and $\mathcal{F}, \mathcal{F}_e \in \mathcal{P}(\mathsf{LState})$

$$\begin{aligned}g \in \mathcal{F} \wedge \mathcal{F} \blacktriangleright \mathcal{I} \wedge s_e \in \mathcal{F}_e \wedge \mathcal{F}_e \blacktriangleright \mathcal{I}_e \wedge \mathcal{I}\downarrow (s, g - s, \mathcal{I}_0) \\ \implies \mathcal{I} \cup \mathcal{I}_e \downarrow (s, (g - s) \circ s_e, \mathcal{I}_0)\end{aligned}$$

Proof. Pick an arbitrary $\mathcal{I}_0, \mathcal{I}, \mathcal{I}_e \in \mathsf{AMod}$, $s, g, s_e \in \mathsf{LState}$ and $\mathcal{F}, \mathcal{F}_e \in \mathcal{P}(\mathsf{LState})$ such that

$$g \in \mathcal{F} \wedge s_e \in \mathcal{F}_e \tag{A.81}$$

$$\mathcal{F} \blacktriangleright \mathcal{I} \wedge \mathcal{F}_e \blacktriangleright \mathcal{I}e \tag{A.82}$$

$$\mathcal{I}\downarrow (s, g - s, \mathcal{I}_0) \tag{A.83}$$

From the definition of $\downarrow$, it then suffices to show

$$\forall n \in \mathbb{N}.\, \mathcal{I} \cup \mathcal{I}_e \downarrow_n (s, (g - s) \circ s_e, \mathcal{I}_0) \tag{A.84}$$

RTS. (A.84)
Rather than proving (A.84) directly, we first establish the following.

$$\begin{aligned}&\forall n \in \mathbb{N}.\, \forall s, g, s_e \in \mathsf{LState}. \\ &\quad g \in \mathcal{F} \wedge s_e \in \mathcal{F}_e \wedge \mathcal{I}\downarrow_n (s, g - s, \mathcal{I}_0) \implies \\ &\quad\quad \mathcal{I} \cup \mathcal{I}_e \downarrow_n (s, (g - s) \circ s_e, \mathcal{I}_0)\end{aligned} \tag{A.85}$$

We can then despatch (A.84) from (A.81)-(A.83) and (A.85); since for an arbitrary $n \in \mathbb{N}$, from (A.83) and the definition of $\downarrow$ we have $\mathcal{I}\downarrow_n (s, g - s, \mathcal{I}_0)$ and consequently from (A.81) and (A.85) we derive $\mathcal{I} \cup \mathcal{I}_e \downarrow_n (s, (g - s) \circ s_e, \mathcal{I}_0)$ as required.

RTS. (A.85)
We proceed by induction on the number of steps n.

Base case $n = 0$
Pick an arbitrary $s, g, s_e \in \mathsf{LState}$. We are then required to show $\mathcal{I} \cup \mathcal{I}_e{\downarrow_0}$ $(s, (g - s) \circ s_e, \mathcal{I}_0)$ which follows trivially from the definition of $\downarrow_0$.

Inductive case
Pick an arbitrary $n \in \mathbb{N}$ and $s, r, g, s_e \in \mathsf{LState}$ such that

$$g \in \mathcal{F} \tag{A.86}$$
$$s_e \in \mathcal{F}_e \tag{A.87}$$
$$g = s \circ r \tag{A.88}$$
$$\mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}_0) \tag{A.89}$$
$$\begin{aligned}&\forall s'', g'', s''_e.\, g'' \in \mathcal{F} \wedge s''_e \in \mathcal{F}_e \wedge \mathcal{I}{\downarrow_{(n-1)}}\ \left(s'', g'' - s'', \mathcal{I}_0\right)\\ &\qquad \implies \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}}\ \left(s'', (g'' - s'') \circ s''_e, \mathcal{I}_0\right)\end{aligned} \tag{I.H}$$

RTS.

$$\forall \kappa.\, \forall a \in \mathcal{I}_0(\kappa).\, (\mathit{potential}(a, g \circ s_e) \Rightarrow \mathit{reflected}(a, g \circ s_e, (\mathcal{I} \cup \mathcal{I}_e)(\kappa)) \tag{A.90}$$

$$\begin{aligned}&\forall \kappa.\, \forall a \in (\mathcal{I} \cup \mathcal{I}_e)\,(\kappa).\, \mathit{potential}(a, g \circ s_e) \Rightarrow\\ &\quad (\mathit{reflected}(a, g \circ s_e, \mathcal{I}_0(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge\\ &\quad \forall (s', r') \in a[s, r \circ s_e].\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}_0)\end{aligned} \tag{A.91}$$

RTS. (A.90)
Pick an arbitrary κ, $a = (p, q, c) \in \mathcal{I}_0(\kappa)$ such that $\mathit{potential}(a, g \circ s_e)$. From (A.88), (A.89) and the definition of $\downarrow_n$ we then have $\mathit{reflected}(a, g, \mathcal{I}(\kappa))$. Consequently, from the definition of *reflected* we trivially have:

$$\mathit{reflected}(a, g, (\mathcal{I} \cup \mathcal{I}_e)(\kappa)) \tag{A.92}$$

Pick an arbitrary $l \in \mathsf{LState}$ such that

$$p \circ c \leq g \circ s_e \circ l \tag{A.93}$$

From (A.92) and the definition of *reflected* we then know there exists a', c' such that

$$a' = (p, q, c') \in (\mathcal{I} \cup \mathcal{I}_e)(\kappa) \wedge p \circ c' \leq g \circ s_e \circ r \tag{A.94}$$

Finally, from (A.93), (A.94) and the definition of *reflected* we have

$$\mathit{reflected}(a, g \circ s_e, (\mathcal{I} \cup \mathcal{I}_e)(\kappa))$$

as required.

RTS. (A.91)
Pick an arbitrary κ, $a = (p, q, c) \in (\mathcal{I} \cup \mathcal{I}_e)(\kappa)$ and (s', r') such that

$$potential(a, g \circ s_e) \tag{A.95}$$
$$(s', r') \in a[s, r \circ s_e] \tag{A.96}$$

Since either $a \in \mathcal{I}(\kappa)$ or $a \in \mathcal{I}_e(\kappa)$, there are two cases to consider:

Case 1. $a \in \mathcal{I}(\kappa)$
Since from (A.95) and the definition of *potential* we have $s_e \circ g \sqcap p \circ c \neq \emptyset$ and consequently, $g \sqcap p \circ c \neq \emptyset$, from (A.86) we have:

$$p \leq g \wedge p \perp s_e \tag{A.97}$$

From (A.95), (A.97) and the definition of *potential* we have:

$$potential(a, g) \tag{A.98}$$

On the other hand, from (A.96), (A.97) and the definitions of $a[s, r \circ s_e]$ and $\perp$, we know there exists r'':

$$r' = r'' \circ s_e \tag{A.99}$$
$$(s', r'') \in a[s, r] \tag{A.100}$$

From (A.100) and the definitions of $a[s, r]$ and $a[s \circ r]$, we know $s' \circ r'' = a[s \circ r]$. Consequently, from (A.82), (A.86), (A.98), the definition of $\blacktriangleright$ and since $g = s \circ r$ (A.88), we have:

$$s' \circ r'' \in \mathcal{F} \tag{A.101}$$

On the other hand, from (A.89), (A.98), (A.95) and (A.100) we have:

$$(reflected(a, s \circ r, \mathcal{I}_0(\kappa)) \vee \neg visible(a, s)) \wedge \tag{A.102}$$
$$\mathcal{I}{\downarrow_{(n-1)}} \left(s', r'', \mathcal{I}_0\right) \tag{A.103}$$

From (A.87), (A.101), (A.102) and (I.H) we have:

$$\mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}} \left(s', r'' \circ s_e, \mathcal{I}_0\right)$$

and thus from (A.99)

$$\mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_0\right) \tag{A.104}$$

Consequently, from (A.96)-(A.104) we have:

$$\forall(s', r') \in a[s, r \circ s_e].\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_0\right) \tag{A.105}$$

From (A.102) there are two cases to consider:
Case 1.1. $\neg \mathit{visible}(a, s)$
From (A.105) and the assumption of the case we have:

$$\begin{aligned} &\neg \mathit{visible}(a, s) \wedge \\ &\forall(s', r') \in a[s, r \circ s_e].\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_0\right) \end{aligned}$$

as required.

Case 1.2. $\mathit{reflected}(a, s \circ r, \mathcal{I}_0(\kappa)$
Pick an arbitrary $l \in \mathsf{LState}$ such that

$$p \circ c \leq g \circ s_e \circ l \tag{A.106}$$

From the assumption of the case, (A.88) and the definition of *reflected* we then have

$$\exists a', c'.\, a' = (p, q, c') \wedge a' \in \mathcal{I}_0(\kappa) \wedge p \circ c' \leq g \circ s_e \circ l \tag{A.107}$$

Thus from (A.106), (A.107) and the definition of *reflected* we have:

$$\mathit{reflected}(a, g \circ s_e, \mathcal{I}_0(\kappa)) \tag{A.108}$$

Thus from (A.105) and (A.108) we have:

$$\begin{aligned} &\mathit{reflected}(a, g \circ s_e, \mathcal{I}_0(\kappa)) \wedge \\ &\forall(s', r') \in a[s, r \circ s_e].\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}} \left(s', r', \mathcal{I}_0\right) \end{aligned}$$

as required.

Case 2. $a \in \mathcal{I}_e(\kappa)$
Since from (A.95) and the definition of *potential* we have $g \circ s_e \sqcap p \circ c \neq \emptyset$ and consequently, $s_e \sqcap p \circ c \neq \emptyset$, from (A.87) and the assumption of the case we have:

$$p \leq s_e \wedge p \perp g \tag{A.109}$$

and consequently from the definition of *visible* and (A.88) we have

$$\neg \mathit{visible}(a, s) \tag{A.110}$$

From (A.95), (A.109) and the definition of *potential* we have:

$$potential(a, s_e) \tag{A.111}$$

From (A.88), (A.109) and the definitions of $a[s, r \circ s_e]$ and $\bot$, we know there exists s'_e such that:

$$s' = s \wedge r' = r \circ s'_e \tag{A.112}$$
$$a[s_e] = s'_e \tag{A.113}$$

Consequently, from (A.82), (A.87), (A.111), (A.113) and the definition of $\blacktriangleright$ we have:

$$s'_e \in \mathcal{F}_e \tag{A.114}$$

From (A.88), (A.112) and Lemma 12 we have:

$$\mathcal{I}{\downarrow_{(n-1)}}\ \left(s', r, \mathcal{I}_0\right) \tag{A.115}$$

From (A.86), (A.114), (A.115), (I.H) we have:

$$\mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}}\ \left(s', r \circ s'_e, \mathcal{I}_0\right)$$

and thus from (A.112)

$$\mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_0\right) \tag{A.116}$$

Finally, from (A.96), (A.110) and (A.116) we have:

$$\begin{array}{l} \neg visible(a, s) \wedge \\ \forall (s', r') \in a[s, r \circ s_e].\, \mathcal{I} \cup \mathcal{I}_e{\downarrow_{(n-1)}}\ \left(s', r', \mathcal{I}_0\right) \end{array}$$

as required.

□

Lemma 12. For all $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ and $n \in \mathbb{N}^+$

$$\forall s, r \in \mathsf{LState}.\, \mathcal{I}{\downarrow_n}\ (s, r, \mathcal{I}') \implies \mathcal{I}{\downarrow_{(n-1)}}\ (s, r, \mathcal{I}')$$

Proof. Pick an arbitrary $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ and proceed by induction on number of steps n.

Base case n = 1
Pick an arbitrary $s, r \in \mathsf{LState}$. We are then required to show $\mathcal{I}{\downarrow_0}\ (s, r, \mathcal{I}')$ which trivially follows from the definition of $\downarrow_0$.

Inductive case
Pick an arbitrary $s, r \in \mathsf{LState}$ such that

$$\mathcal{I}{\downarrow_{(n+1)}}\ (s, r, \mathcal{I}') \tag{A.117}$$
$$\forall s', r' \in \mathsf{LState}.\, \mathcal{I}{\downarrow_n}\ (s', r', \mathcal{I}') \implies \mathcal{I}{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}') \tag{I.H}$$

RTS.

$$\forall \kappa.\, \forall a \in \mathcal{I}'(\kappa).\, \mathit{potential}(a, s \circ r) \Rightarrow \mathit{reflected}(a, s \circ r, \mathcal{I}') \tag{A.118}$$
$$\begin{aligned}&\forall \kappa.\, \forall a \in \mathcal{I}(\kappa).\, \mathit{potential}(a, s \circ r) \Rightarrow \\ &\quad (\mathit{reflected}(a, s \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge \\ &\quad \forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}')\end{aligned} \tag{A.119}$$

RTS. (A.118)
Pick an arbitrary κ and $a \in \mathcal{I}'(\kappa)$ such that $\mathit{potential}(a, s \circ r)$. From (A.117) and the definition of $\downarrow_{(n+1)}$ we then have $\mathit{reflected}(a, s \circ r, \mathcal{I}(\kappa))$ as required.

RTS. (A.119)
Pick an arbitrary κ and $a \in \mathcal{I}(\kappa)$ such that $\mathit{potential}(a, s \circ r)$. Then from (A.117) we have:

$$\begin{aligned}&(\mathit{reflected}(a, s \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge \\ &\forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_n}\ (s', r', \mathcal{I}')\end{aligned}$$

and consequently from (I.H)

$$\begin{aligned}&(\mathit{reflected}(a, s \circ r, \mathcal{I}'(\kappa)) \vee \neg \mathit{visible}(a, s)\ ) \wedge \\ &\forall (s', r') \in a[s, r].\, \mathcal{I}{\downarrow_{(n-1)}}\ (s', r', \mathcal{I}')\end{aligned}$$

as required.

□

Lemma 13 (Sequential command soundness)**.** For all $\mathbb{S} \in \mathsf{Seqs}$, $(M_1, \mathbb{S}, M_2) \in \mathrm{Ax}_{\mathsf{S}}$ and $m \in \mathbb{M}$:

$$[\![\mathbb{S}]\!]_{\mathsf{S}} (\lfloor M_1 \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \subseteq \lfloor M_2 \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}$$

Proof. By induction over the structure of $\mathbb{S}$. Pick an arbitrary $m \in \mathbb{M}$.

Case $\mathbb{E}$
This follows immediately from parameter 10.

Case skip
RTS.

$$[\![\mathbb{S}]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \subseteq \lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}$$

Proof.

$$\begin{array}{rcl} [\![\mathbb{S}]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) = & \lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \\ \subseteq & \lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \end{array}$$

as required.

Case $\mathbb{S}_1; \mathbb{S}_2$
RTS.

$$[\![\mathbb{S}_1; \mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \subseteq \left\lfloor M' \bullet_{\mathbb{M}} \{m\} \right\rfloor_{\mathbb{M}}$$

where $(M, \mathbb{S}_1, M''), (M'', \mathbb{S}_2, M') \in \mathrm{Ax}_{\mathsf{S}}$.

Proof.

$$\begin{array}{rcl} [\![\mathbb{S}_1; \mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) = & [\![\mathbb{S}_2]\!]_{\mathsf{S}} ([\![\mathbb{S}_1]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}})) \\ \text{(I.H.)} \quad \subseteq & [\![\mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M'' \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \\ \text{(I.H.)} \quad \subseteq & \lfloor M' \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \end{array}$$

as required.

Case $\mathbb{S}_1 + \mathbb{S}_2$
RTS.

$$[\![\mathbb{S}_1 + \mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \subseteq \left\lfloor M' \bullet_{\mathbb{M}} \{m\} \right\rfloor_{\mathbb{M}}$$

where $(M, \mathbb{S}_1, M'), (M, \mathbb{S}_2, M') \in \mathrm{Ax}_{\mathsf{S}}$.

Proof.

$$\begin{array}{rcl} [\![\mathbb{S}_1 + \mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) = & [\![\mathbb{S}_1]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \cup [\![\mathbb{S}_2]\!]_{\mathsf{S}} (\lfloor M \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}}) \\ \text{(I.H.)} \quad \subseteq & \lfloor M' \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \cup \lfloor M' \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \\ \subseteq & \lfloor M' \bullet_{\mathbb{M}} \{m\} \rfloor_{\mathbb{M}} \end{array}$$

as required.

Case $\mathbb{S}^\star$
RTS.

$$[\![\mathbb{S}^\star]\!]_{\mathsf{S}}\,(\lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}) \subseteq \lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}$$

where $(M, \mathbb{S}, M) \in \mathrm{Ax}_{\mathsf{S}}$.

Proof.

$$\begin{array}{rll} [\![\mathbb{S}^\star]\!]_{\mathsf{S}}\,(\lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}) = & [\![\,\mathrm{skip} + \mathbb{S}; \mathbb{S}^\star]\!]_{\mathsf{S}}\,(\lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}) \\ = & [\![\,\mathrm{skip}]\!]_{\mathsf{S}}\,(\lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}) \cup [\![\mathbb{S}; \mathbb{S}^\star]\!]_{\mathsf{S}}\,(\lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}}) \\ \text{(I.H.)} \quad \subseteq & \lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}} \cup \lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}} \\ \subseteq & \lfloor M \bullet_{\mathbb{M}} \{m\}\rfloor_{\mathbb{M}} \end{array}$$

as required.

□

Lemma 14. For all $w_1, w_2, w, w' = (l', g', \mathcal{I}') \in \mathsf{World}$,

$$w_1 \bullet w_2 = w \wedge (l', g', \mathcal{I}') \in G(w_1) \implies ((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R(w_2)$$

Proof. Pick an arbitrary w_1, w_2, w such that:

$$w_1 \bullet w_2 = w \tag{A.120}$$
$$(l', g', \mathcal{I}') \in G(w_1) \tag{A.121}$$

RTS.

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R(w_2)$$

From (A.121) and by definition of G we know:

$$(l', g', \mathcal{I}') \in (G^{\mathsf{u}} \cup G^{\mathsf{e}})^* (w_2) \tag{A.122}$$

From (A.120), (A.122) and by Lemmata 15 and 16 we have:

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in (R^{\mathsf{u}} \cup R^{\mathsf{e}})^* (w_2)$$

and consequently

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R(w_2)$$

as required. □

Lemma 15. For all $w_1, w_2, w, w' = (l', g', \mathcal{I}') \in \mathsf{World}$,

$$w_1 \bullet w_2 = w \wedge (l', g', \mathcal{I}') \in G^{\mathsf{u}}(w_1) \implies ((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{u}}(w_2)$$

where we write $R^{\mathsf{u}}(w)$ to denote $\{w' \mid (w, w') \in R^{\mathsf{u}}(w)\}$.

Proof. Pick an arbitrary $W_1 = (l_1, g_1, \mathcal{I}_1)$, $w_2 = (l_2, g_2, \mathcal{I}_2), w$ and $(l', g', \mathcal{I}')$ such that:

$$w_1 \bullet w_2 = w \tag{A.123}$$
$$(l', g', \mathcal{I}') \in G^{\mathsf{u}}(w_1) \tag{A.124}$$

RTS.

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{u}}(w_2)$$

* From (A.123) we know:

$$g_1 = g_2 \tag{A.125}$$
$$\mathcal{I}_1 = \mathcal{I}_2 \tag{A.126}$$

By definition of $G^{\mathsf{u}}$and from (A.124) and (A.126) we know:

$$\mathcal{I}' = \mathcal{I}_1 = \mathcal{I}_2 \tag{A.127}$$
$$(l_1 \circ g_1)_{\mathsf{K}}^{\perp}{}_{\mathbb{K}} = \left((l' \circ g')_{\mathsf{M}}\right)^{\perp}_{\mathbb{K}} \tag{A.128}$$
$$g' = g_1 \vee \left(\begin{array}{l} \exists \kappa \leq (l_1)_{\mathsf{K}}.\, (g_1, g') \in \lceil \mathcal{I}_1 \rceil\, (\kappa) \wedge \\ ((l_1 \circ g_1)_{\mathsf{M}})^{\perp}_{\mathbb{M}} = ((l' \circ g')_{\mathsf{M}})^{\perp}_{\mathbb{M}} \end{array} \right)$$

There are two cases to consider:

Case 1. $g_1 = g'$
From (A.125) and the assumption of the case we know $g' = g_2$. Consequently, from (A.127) we have:

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') = (l_2, g_2, \mathcal{I}_2) \tag{A.129}$$

By definition of R^{u} and from (A.129) we can conclude:

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{u}}(l_2, g_2, \mathcal{I}_2) \tag{A.130}$$

as required.

Case 2.

$$\exists \kappa \leq (l_1)_{\mathsf{K}}.\, (g_1, g') \in \lceil \mathcal{I}_1 \rceil\, (\kappa) \tag{A.131}$$
$$((l_1 \circ g_1)_{\mathsf{M}})^{\perp}_{\mathbb{M}} = \left((l' \circ g')_{\mathsf{M}}\right)^{\perp}_{\mathbb{M}} \tag{A.132}$$

From (A.123), (A.125) and (A.126) we know that

$$w = (l_1 \circ l_2, g_2, \mathcal{I}_2) \tag{A.133}$$

Since $\mathsf{wf}\,(w)$ (by definition of World) and from (A.125) we know:

$$(l_1 \circ l_2 \circ g_2)_{\mathsf{K}} = (l_1)_{\mathsf{K}} \bullet_{\mathbb{K}} (l_2)_{\mathsf{K}} \bullet_{\mathbb{K}} (g_2) = (l_1 \circ g_1)_{\mathsf{K}} \bullet_{\mathbb{K}} (l_2)_{\mathsf{K}} \text{ is defined} \tag{A.134}$$
$$(l_1 \circ l_2 \circ g_1)_{\mathsf{M}} = (l_1 \bullet_{\mathbb{M}} g_1)_{\mathsf{M}} \bullet_{\mathbb{M}} (l_2)_{\mathsf{M}} \text{ is defined} \tag{A.135}$$

Since $\kappa_1 \leq (l_1)_{\mathsf{K}}$ (A.131), from (A.134) and Lemma 20, we know:

$$\kappa \sharp \ (l_2)_{\mathsf{K}} \bullet_{\mathbb{K}} (g_2)_{\mathsf{K}} \tag{A.136}$$

From (A.125), (A.132) and (A.135) we know

$$(l' \circ g')_{\mathsf{M}} \bullet_{\mathbb{M}} (l_2)_{\mathsf{M}} = (l' \circ l_2 \circ g')_{\mathsf{M}} \quad \text{is defined} \tag{A.137}$$

From (A.128) and (A.134) we know

$$(l' \circ g')_{\mathsf{K}} \bullet_{\mathbb{K}} (l_2)_{\mathsf{K}} = (l' \circ l_2 \circ g')_{\mathsf{K}} \quad \text{is defined} \tag{A.138}$$

From (A.137) and (A.138) we know $l'_1 \circ l_2 \circ g'$ is defined and consequently:

$$l_2 \circ g' \quad \text{is defined} \tag{A.139}$$

From (A.126), (A.131), (A.136), (A.139) and by definition of R^{u}, we have:

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') = (l_2, g_2, \mathcal{I}_2) \in R^{\mathsf{u}}(l_2, s_2, \mathcal{I}_2)$$

as required. □

Lemma 16. For all $w_1, w_2, w, w' = (l', g', \mathcal{I}') \in \mathsf{World}$,

$$w_1 \bullet w_2 = w \wedge w' \in G^{\mathsf{e}}(w_1) \implies ((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{e}}(w_2)$$

Proof. Pick an arbitrary $w_1 = (l_1, g_1, \mathcal{I}_1)$, $w_2 = (l_2, g_2, \mathcal{I}_2), w$ and $w' = (l', g', \mathcal{I}')$ such that:

$$w_1 \bullet w_2 = w \tag{A.140}$$

$$w' \in G^{\mathsf{e}}(w_1) \tag{A.141}$$

RTS.

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{e}}(w_2)$$

From (A.140) we know:

$$g_1 = g_2 \wedge \mathcal{I}_1 = \mathcal{I}_2 \tag{A.142}$$

By definition of G^{e} and from (A.141) and (A.142) we know there exists $l_3, l_4, g'' \in \mathsf{LState}$, $\kappa_1, \kappa_2 \in \mathbb{K}$ and $\mathcal{I}''$ such that

$$\begin{array}{l} l_1 = l_3 \circ l_4 \wedge l'_1 = l_3 \circ (\mathbf{0}_{\mathbb{M}}, \kappa_1) \\ \wedge g'' = l_4 \circ (\mathbf{0}_{\mathbb{M}}, \kappa_2) \wedge g' = g_2 \circ g'' \\ \wedge \kappa_1 \bullet_{\mathbb{K}} \kappa_2 \prec \mathsf{dom}\,(\mathcal{I}'') \wedge \kappa_1 \bullet_{\mathbb{K}} \kappa_2 \perp \mathsf{dom}\,(\mathcal{I}_2) \\ \wedge \mathcal{I}' = \mathcal{I}_2 \cup \mathcal{I}'' \wedge \mathcal{I}'' \uparrow^{(g'')} (g_2, \mathcal{I}_2) \wedge g'' \circledcirc \mathcal{I}'' \end{array} \tag{A.143}$$

From (A.143) and the definition of R^{e} we have:

$$((w_2)_{\mathsf{L}}, g', \mathcal{I}') \in R^{\mathsf{e}}(w_2)$$

as required. □

Lemma 17. for all $P \in \mathsf{Assn}$, $\iota \in \mathsf{LEnv}$, $w, w' \in \mathsf{World}$,

$$w, \iota \vDash P \wedge (w, w') \in R^{\mathsf{e}} \implies w', \iota \vDash P$$

Proof. From the definition of R^{e}, it then suffices to show that for all $P \in \mathsf{Assn}$, $\iota \in \mathsf{LEnv}$, $l, g, g' \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$:

$$(l, g, \mathcal{I}), \iota \vDash P \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \implies (l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P$$

We proceed by induction on the structure of assertion P.

Case $P \stackrel{\text{def}}{=} A$ Immediate.
Case $P \stackrel{\text{def}}{=} P_1 \implies P_2$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $l, g, g' \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ such that

$$(l, g, \mathcal{I}), \iota \vDash P_1 \vee P_2 \tag{A.144}$$
$$\mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \tag{A.145}$$
$$\forall l, g, g' \in \mathsf{LState}.\, \forall \mathcal{I}', \mathcal{I} \in \mathsf{AMod}.$$
$$(l, g, \mathcal{I}), \iota \vDash P_1 \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \implies (l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P_1 \tag{I.H1}$$
$$\forall l, g, g' \in \mathsf{LState}.\, \forall \mathcal{I}', \mathcal{I} \in \mathsf{AMod}.$$
$$(l, g, \mathcal{I}), \iota \vDash P_2 \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \implies (l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P_2 \tag{I.H2}$$

From (A.144) and the definition of $\vDash$ we know $(l, g, \mathcal{I}), \iota \vDash P_1$ or $(l, g, \mathcal{I}), \iota \vDash P_2$; consequently, from (A.145), (I.H1) and (I.H2) we have: $(l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P_1$ or $(l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P_2$. Thus, from the definition of $\vDash$ we have:

$$(l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P_1 \vee P_2$$

as required.

Cases $P \stackrel{\text{def}}{=} \exists x.\, P'$**;** $P \stackrel{\text{def}}{=} P_1 \ast P_2$**;** $P \stackrel{\text{def}}{=} P_1 \mathbin{⫤} P_2$**;** $P \stackrel{\text{def}}{=} P_1 \wedge P_2$
These cases are analogous to the previous case and are omitted here.

Case $P \stackrel{\text{def}}{=} \boxed{P'}_I$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ such that

$$(l, g, \mathcal{I}), \iota \vDash \boxed{P'}_I \tag{A.146}$$
$$\mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \tag{A.147}$$
$$\forall l, g, g' \in \mathsf{LState}.\, \forall \mathcal{I}', \mathcal{I} \in \mathsf{AMod}.$$
$$(l, g, \mathcal{I}), \iota \vDash P' \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \implies (l, g \circ g', \mathcal{I} \cup \mathcal{I}'), \iota \vDash P' \tag{I.H}$$

From (A.146) and the definition of $\vDash$ we have:

$$l = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g = s' \circ r' \wedge s', \iota \vDash_{g,\mathcal{J}} P' \wedge \mathcal{J}{\downarrow}\left(s', r', \langle\!|I|\!\rangle_{\iota}\right)$$

Thus from (A.147) and Lemma 18 we have

$$l = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g \circ g' = s' \circ r' \circ g' \wedge s', \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P' \wedge \mathcal{J}{\downarrow}\left(s', r', \langle\!|I|\!\rangle_{\iota}\right)$$

and consequently from (A.147) and (I.H) we have

$$l = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g \circ g' = s' \circ r' \wedge s', \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P' \wedge \mathcal{J}\cup\mathcal{J}'{\downarrow}\left(s', r', \langle\!|I|\!\rangle_{\iota}\right)$$

That is,

$$(l, g \circ g', \mathcal{J} \cup \mathcal{J}'), \iota \vDash \boxed{P'}_{I} \tag{A.148}$$

as required. □

Lemma 18. for all $P \in \mathsf{Assn}$, $\iota \in \mathsf{LEnv}$, $s, g, g' \in \mathsf{LState}$ and $\mathcal{J}, \mathcal{J}' \in \mathsf{AMod}$:

$$s, \iota \vDash_{g,\mathcal{J}} P \wedge \mathcal{J}'{\uparrow}^{(g')}\,(g, \mathcal{J}) \implies s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P$$

Proof. By induction on the structure of assertion P.

Case $P \stackrel{\text{def}}{=} A$ Immediate.
Case $P \stackrel{\text{def}}{=} P_1 \implies P_2$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g, g' \in \mathsf{LState}$ and $\mathcal{J}, \mathcal{J}' \in \mathsf{AMod}$ such that

$$s, \iota \vDash_{g,\mathcal{J}} P_1 \implies P_2 \tag{A.149}$$
$$\mathcal{J}'{\uparrow}^{(g')}\,(g, \mathcal{J}) \tag{A.150}$$
$$\forall s, g, g' \in \mathsf{LState}.\, \forall \mathcal{J}', \mathcal{J} \in \mathsf{AMod}.\quad s, \iota \vDash_{g,\mathcal{J}} P_1 \wedge \mathcal{J}'{\uparrow}^{(g')}\,(g, \mathcal{J}) \implies s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P_1 \tag{I.H1}$$
$$\forall s, g, g' \in \mathsf{LState}.\, \forall \mathcal{J}', \mathcal{J} \in \mathsf{AMod}.\quad s, \iota \vDash_{g,\mathcal{J}} P_2 \wedge \mathcal{J}'{\uparrow}^{(g')}\,(g, \mathcal{J}) \implies s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P_2 \tag{I.H2}$$

From (A.149) and the definition of $\vDash_{g,\mathcal{J}}$ we know $s, \iota \vDash_{g,\mathcal{J}} P_1$ implies $P_2 \vDash_{g,\mathcal{J}}$; consequently, from (A.150), (I.H1) and (I.H2) we have: $s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P_1$ implies $s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P_2$. Thus, from the definition of $\vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'}$ we have:

$$s, \iota \vDash_{g\circ g', \mathcal{J}\cup\mathcal{J}'} P_1 \implies P_2$$

as required.

Cases $P \stackrel{\text{def}}{=} \exists x.\, P'$**;** $P \stackrel{\text{def}}{=} P_1 \ast P_2$**;** $P \stackrel{\text{def}}{=} P_1 \mathbin{\text{-}\!\!\ast} P_2$
These cases are analogous to the previous case and are omitted here.

Case $P \stackrel{\text{def}}{=} \boxed{P'}_I$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ such that

$$s, \iota \vDash_{g,\mathcal{I}} \boxed{P'}_I \tag{A.151}$$

$$\mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \tag{A.152}$$

$$\forall s, g, g' \in \mathsf{LState}.\, \forall \mathcal{I}', \mathcal{I} \in \mathsf{AMod}.$$
$$s, \iota \vDash_{g,\mathcal{I}} P' \wedge \mathcal{I}' \uparrow^{(g')} (g, \mathcal{I}) \implies s, \iota \vDash_{g \circ g', \mathcal{I} \cup \mathcal{I}'} P' \tag{I.H}$$

From (A.151) and the definition of $\vDash_{g,\mathcal{I}}$ we have:

$$s = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g = s' \circ r' \wedge s', \iota \vDash_{g,\mathcal{I}} P' \wedge \mathcal{I} \downarrow \left(s', r', \langle\!| I |\!\rangle_\iota\right)$$

Thus from (A.152) and (I.H) we have

$$s = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g \circ g' = s' \circ r' \wedge s', \iota \vDash_{g \circ g', \mathcal{I} \cup \mathcal{I}'} P' \wedge$$
$$\mathcal{I} \cup \mathcal{I}' \downarrow \left(s', r', \langle\!| I |\!\rangle_\iota\right)$$

That is,

$$s, \iota \vDash_{g, \mathcal{I} \cup \mathcal{I}'} \boxed{P'}_I$$

as required. □

Lemma 19. for all $P \in \mathsf{Assn}$, $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$:

$$(s, g, \mathcal{I}), \iota \vDash P \wedge \mathcal{I}' \uparrow^{(s)} (g, \mathcal{I}) \implies s, \iota \vDash_{g \circ s, \mathcal{I} \cup \mathcal{I}'} P$$

Proof. By induction on the structure of assertion P.

Case $P \stackrel{\text{def}}{=} A$ Immediate.
Case $P \stackrel{\text{def}}{=} P_1 \implies P_2$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathcal{I}, \mathcal{I}' \in \mathsf{AMod}$ such that

$$(s, g, \mathcal{I}), \iota \vDash P_1 \implies P_2 \tag{A.153}$$

$$\mathcal{I}' \uparrow^{(s)} (g, \mathcal{I}) \tag{A.154}$$

$$\forall s, g \in \mathsf{LState}.\, \forall \mathcal{I}, \mathcal{I}' \in \mathsf{AMod}.$$

$$(s, g, \mathfrak{I}), \iota \vDash P_1 \wedge \mathfrak{I}' \uparrow^{(s)} (g, \mathfrak{I}) \implies s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P_1 \tag{I.H1}$$

$$\forall s, g \in \mathsf{LState}.\, \forall \mathfrak{I}, \mathfrak{I}' \in \mathsf{AMod}.$$

$$(s, g, \mathfrak{I}), \iota \vDash P_2 \wedge \mathfrak{I}' \uparrow^{(s)} (g, \mathfrak{I}) \implies s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P_2 \tag{I.H2}$$

From (A.153) and the definition of $\vDash$ we know $(s, g, \mathfrak{I}), \iota \vDash P_1$ implies $(s, g, \mathfrak{I}), \iota \vDash P_2$; consequently, from (A.154), (I.H1) and (I.H2) we have: $s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P_1$ implies $s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P_2$. Thus, from the definition of $\vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'}$ we have:

$$s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P_1 \implies P_2$$

as required.

Cases $P \stackrel{\text{def}}{=} \exists x.\, P'$**;** $P \stackrel{\text{def}}{=} P_1 \star P_2$**;** $P \stackrel{\text{def}}{=} P_1 \mathbin{\text{⍟}} P_2$
These cases are analogous to the previous case and are omitted here.

Case $P \stackrel{\text{def}}{=} \boxed{P'}_I$
Pick an arbitrary $\iota \in \mathsf{LEnv}$, $s, g \in \mathsf{LState}$ and $\mathfrak{I}, \mathfrak{I}' \in \mathsf{AMod}$ such that

$$(s, g, \mathfrak{I}), \iota \vDash \boxed{P'}_I \tag{A.155}$$

$$\mathfrak{I}' \uparrow^{(s)} (g, \mathfrak{I}) \tag{A.156}$$

From (A.155) and the definition of $\vDash$ we have:

$$s = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g = s' \circ r' \wedge s', \iota \vDash_{g, \mathfrak{I}} P' \wedge \mathfrak{I} \downarrow \left(s', r', \langle\!| I |\!\rangle_\iota\right)$$

Thus from (A.156) we have

$$s = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g = s' \circ r' \wedge s', \iota \vDash_{g, \mathfrak{I}} P' \wedge \mathfrak{I} \cup \mathfrak{I}' \downarrow \left(s', r' \circ s, \langle\!| I |\!\rangle_\iota\right)$$

and consequently from (A.156), Lemma 18 and since $s = \mathbf{0}_{\mathsf{L}}$

$$\begin{aligned} &s = \mathbf{0}_{\mathsf{L}} \wedge \exists s', r'.\, g \circ s = s' \circ r' \wedge s', \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} P' \\ &\wedge \mathfrak{I} \cup \mathfrak{I}' \downarrow \left(s', r' \circ s, \langle\!| I |\!\rangle_\iota\right) \end{aligned}$$

That is,

$$s, \iota \vDash_{g \circ s, \mathfrak{I} \cup \mathfrak{I}'} \boxed{P'}_I$$

as required. □

Lemma 20. Given any separation algebra $(\mathcal{M}, \bullet_{\mathcal{M}}, \mathbf{0}_{\mathcal{M}})$,

$$\forall a, b, c, d \in \mathcal{M}.\, a \bullet_{\mathcal{M}} b = d \ \wedge\ c \leq b \implies \exists f \in \mathcal{M}.\, a \bullet_{\mathcal{M}} c = f$$

Proof. Pick an arbitrary $a, b, c, d \in \mathcal{M}$ such that:

$$a \bullet_{\mathcal{M}} b = d \tag{A.157}$$
$$c \leq b \tag{A.158}$$

From (A.158), we have:

$$\exists e \in \mathcal{M}.\, c \bullet_{\mathcal{M}} e = b \tag{A.159}$$

and consequently from (A.157) we have:

$$a \bullet_{\mathcal{M}} c \bullet_{\mathcal{M}} e = d \tag{A.160}$$

Since $e \leq d$ (A.160), we have:

$$\exists f \in \mathcal{M}.\, e \bullet_{\mathcal{M}} f = d \tag{A.161}$$

From (A.160), (A.161) and cancellativity of separation algebras we have:

$$a \bullet_{\mathcal{M}} c = f \tag{A.162}$$

and thus

$$\exists f \in \mathcal{M}.\, a \bullet_{\mathcal{M}} c = f \tag{A.163}$$

as required. □