Manifest Sharing with Session Types

Stephanie Balzer and Frank Pfenning

March 2017
CMU-CS-17-106

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work supported by a Mozilla Research grant and partially sponsored by NSF Grant CNS-1423168: "Blameworthy Programs: Accountability via Deviance and Causal Determination". Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Mozilla Research or NSF.

Keywords: Linear logic, Curry-Howard isomorphism, session types, message-passing concurrency, sharing

Abstract

Session-typed languages building on the Curry-Howard isomorphism between linear logic and session-typed communication guarantee session fidelity and deadlock freedom. Unfortunately, these strong guarantees exclude many naturally occurring programming patterns pertaining to shared resources. In this paper, we introduce sharing into a session-typed language where types are stratified into linear and shared layers with modal operators connecting the layers. The resulting language retains session fidelity but not the absence of deadlocks, which can arise from contention for shared processes. We illustrate our language on various examples, such as the dining philosophers problem, and show that sharing recovers the computational power of the untyped asynchronous π-calculus.

Contents

1 Introduction

Session types [29, 30, 31] prescribe the communication protocols that arise in concurrent programming. Session types and session type libraries have found their ways into various practical programming languages [18, 33, 34, 44, 53] to express such protocols and ensure their adherence at compile-time. Recently, message-passing concurrency has been put onto a firm logical foundation by exhibiting a Curry-Howard isomorphism between linear logic and session-typed communication [8, 9, 57, 62]. Programming languages [26, 58] based on this isomorphism not only guarantee *session fidelity* (preservation) but also a form of *global progress*, since the process graph forms a tree and is acyclic by construction.

Unfortunately, these strong guarantees preclude programming scenarios that naturally demand sharing, such as shared databases or output devices, or implementations that make use of sharing for performance considerations. The shared channels available through the exponential modality in linear logic have a copying semantics [8, 62] and therefore do not provide the correct tools in such applications. In this paper, *shared channels* and *shared processes* always refer to mutable resources.

In this paper, we contribute a session-typed programming language for message-passing concurrency that seamlessly integrates *linear* and *shared* processes. The language allows multiple *aliases* to a shared process to exist, but makes sure that any state-altering communication with such a process only happens once exclusive access to the process has been obtained. At this point, the process becomes linear and can become shared again once it is released, resulting in renunciation of exclusive access. The resulting language retains session fidelity but not the absence of deadlocks, which can arise from contention for shared processes.

A key novelty of our work is to go beyond supporting *acquire-release* as a mere language primitive, but to enrich the type system so that a session type prescribes at which points in the protocol acquisition and release must happen. We generalize the idea of type *stratification* introduced in [49], based on Benton's LNL 1994 and Reed's adjoint logic 2009, and stratify session types into a linear and shared layer and support two *modalities* going back and forth between them. We then interpret the modal operator shifting *down* from the shared to the linear layer as a *release* and the operator shifting *up* from the linear to the shared layer as an *acquire*. As a result, we obtain a type system where any form of synchronization, including the acquisition and release of a shared process, is manifest in the session type.

Now that types prescribe the acquisition and release points of shared processes, it is only a small step to making sure that the assumptions by a client attempting to acquire a shared process are actually met. When there is contention for a shared process and one client obtains access at type A and then releases it again, the release must happen again at the same type A. This is necessary since the acquire/release cycle is invisible to all other clients. To capture this constraint statically we introduce the notion of an *equi-synchronizing* session type. A session type is equi-synchronizing if it satisfies the invariant that any release restores the session to the same type at which a preceding acquire occurred.

We illustrate our language on various examples, such as producer-consumer queues and dining philosophers, and also demonstrate how nondeterministic choice can be emulated in the resulting language thanks to shared processes. Moreover, we provide an encoding of the untyped asynchronous π-calculus into our language, recovering the computational power of the untyped π-calculus for session-typed, message-passing concurrency.

An interesting question is what the meta-theoretic consequences of the introduction of sharing are. The correspondence between linear logic and session-typed communication [8, 9, 57, 62] established for purely linear session-typed languages seems no longer to hold in its original form. Under this interpretation proofs correspond to processes and cut reduction to communication. With the introduction of sharing, on the other hand, shared channels upon which a process depends may not always be available. Such a computation state corresponds to an *incomplete proof*. Overall, computation is then an interleaving of *proof construction* (acquiring a resource), *proof reduction* (communication), and *proof deconstruction* (releasing a resource). The fact that computation may deadlock is always a failure of proof construction, never communication.

The principal contributions of this paper are:

- the introduction of sharing into session-typed, message-passing, concurrent programming such that sharing is manifest in the type structure via adjoint modalities;
- its elaboration in the programming language $\mathsf{SILL_S}$, resulting in type system, synchronous operational semantics,

and proofs of session fidelity (preservation) and a modified form of progress that characterizes possible deadlocks;
- the notion of an equi-synchronizing session type to guarantee session fidelity without the need for run-time type checking when acquiring a process;
- an encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$, reclaiming the expressiveness of the untyped π-calculus;
- an extension of the formal system to accommodate an asynchronous dynamics, using a novel transformation derived from logic;
- a prototype implementation of manifest sharing in Concurrent C0.

The main part of this paper is structured as follows: Section 2 provides a brief introduction to linear session types. Section 3 introduces manifest sharing. Section 4 illustrates manifest sharing on various examples. Section 5 details the semantics of $\mathsf{SILL_S}$, including preservation and progress. Section 6 gives the encoding of the untyped asynchronous π-calculus into $\mathsf{SILL_S}$. Section 7 provides a brief overview of our implementation. Section 8 summarizes the related work, and Section 9 concludes the main part of this paper with a discussion and some remarks about future work. The appendix gives the complete statics and dynamics of $\mathsf{SILL_S}$, proofs of preservation and progress as well as of supporting lemmas and corollaries, and further examples.

2 Background

In this section, we provide a short introduction to linear session-typed message-passing concurrency based on the functional language SILL [26, 49, 58] built on the Curry-Howard isomorphism between intuitionistic linear logic and session-typed concurrency. SILL incorporates processes into a functional core via a linear contextual monad that isolates session-typed concurrency. In this introduction we focus on the linear process layer of SILL, which we extend with manifest sharing in Section 3.

Linear logic [24] is a substructural logic that restricts the structural rules of weakening and contraction to propositions of the form $!A$, where $!$ is a so-called exponential modality. As result, purely linear propositions (that is, propositions without an exponential modality) can be viewed as resources that must be used *exactly once* in a proof. We adopt the intuitionistic version of linear logic, which yields the following sequent [13]

$$A_1, \ldots, A_n \vdash A$$

where $A_1, \ldots, A_n$ are linear antecedents and A is the succedent.

Under the Curry-Howard isomorphism for intuitionistic linear logic, propositions are related to session types, proofs to processes, and cut reduction in proofs to communication. Appealing to this correspondence, we assign a process term P to the above judgment and label each hypothesis as well as the conclusion with a *channel*:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

The resulting judgment states that process P *provides* a service of session type A along channel x, *using* the services of session types $A_1, \ldots, A_n$ provided along channels $x_1, \ldots, x_n$. The assignment of a channel to the conclusion is convenient because, unlike functions, processes do not evaluate to a value but continue to communicate along their providing channel once they have been created. For the judgment to be well-formed, all the channel names have to be distinct. In particular, the channel name to the right of the turnstile cannot appear to its left. This intuitionistic interpretation of linear logic avoids the need for explicit dualization [29, 30, 62] of a session type. Whether a session type is used or provided is determined by its positioning to the left or right, respectively, of the turnstile.

The balance between providing and using a session is established by the two fundamental rules of the sequent calculus that are independent of all logical connectives: cut and identity. Cut states that if P provides service A along channel x, then Q can use the service along the same channel at the same type. Identity states that, if we are a client of a service A we can always directly provide A.

$$\frac{\Delta \vdash P_x :: (x : A) \quad \Delta', x : A \vdash Q_x :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P_x \,;\, Q_x :: (z : C)}\ (\text{T-Cut}) \qquad \frac{}{y : A \vdash \mathsf{fwd}\ x\ y :: (x : A)}\ (\text{T-Id})$$

Operationally, the process $x \leftarrow P_x \,; Q_x$ creates a globally fresh channel c, spawns a new process $[c/x]P_x$ providing along c, and continues as $[c/x]Q_x$. Conversely, the process $\mathsf{fwd}\ c\ d$ terminates after directly identifying channels c and d. Here, we have adopted the convention to use x, y, and z for channel *variables* and c and d for *channels*. Channels are created at run-time and substituted for channel variables in process terms. For type-checking purposes, the distinction between variables and channels is negligible, even though they obey slightly different invariants (see Section B.2); we hence use the metavariabels x, y, and z to stand for either of them when type-checking process terms.

The Curry-Howard correspondence gives each connective of linear logic an interpretation as a session type. This session type prescribes the kind of message that must be sent or received along a channel of this type and at which type the session continues after the exchange. Table 1 provides an overview of the session types arising from linear logic and their operational meaning. We generalize internal $A \oplus B$ and external choice $A \mathbin{\&} B$ to n-ary labeled choices $\oplus\{\overline{l : A}\}$ and $\&\{\overline{l : A}\}$, respectively, where we use the overline-notation to denote a sequence, as is usual. We require external and internal choice to comprise at least one label. Otherwise, there would exist a linear channel without observable interaction along it, which is computationally not really interesting and furthermore would complicate our proofs. Because we adopt the intuitionistic version of linear logic, session types are expressed from the point of view of the provider. In the first line of each connective, Table 1 provides the point of view of the *provider* and in the second line the point of view of the *client*. For each connective, its session type before the exchange (**Session type current**) and after the exchange (**Session type continuation**) is given. Likewise, the implementing process term is indicated before the exchange (**Process term current**) and after the exchange (**Process term continuation**). Table 1 shows that the process terms of a provider and a client for a connective come in matching pairs. Both participants' view of the session changes consistently. The process typing rules for the connectives shown in Table 1 can be found in Figure 3. We defer the discussion of the process typing judgment to Section 3.2.

Session type		**Process term**		
current	**continuation**	**current**	**continuation**	**Description**
$c : \oplus\{\overline{l : A}\}$	$c : A_h$	$c.l_h \,; P$	P	provider sends label l_h along c
		$\mathsf{case}\ c\ \mathsf{of}\ \overline{l \Rightarrow Q}$	Q_h	client receives label l_h along c
$c : \&\{\overline{l : A}\}$	$c : A_h$	$\mathsf{case}\ c\ \mathsf{of}\ \overline{l \Rightarrow P}$	P_h	provider receives label l_h along c
		$c.l_h \,; Q$	Q	client sends label l_h along c
$c : A \otimes B$	$c : B$	$\mathsf{send}\ c\ d \,; P$	P	provider sends channel $d : A$ along c
		$y \leftarrow \mathsf{recv}\ c \,; Q_y$	$[d/y]\, Q_y$	client receives channel $d : A$ along c
$c : A \multimap B$	$c : B$	$y \leftarrow \mathsf{recv}\ c \,; P_y$	$[d/y]\, P_y$	provider receives channel $d : A$ along c
		$\mathsf{send}\ c\ d \,; Q$	Q	client sends channel $d : A$ along c
$c : \mathbf{1}$	-	$\mathsf{close}\ c$	-	provider sends "end" along c
		$\mathsf{wait}\ c \,; Q$	Q	provider receives "end" along c

Table 1: Overview of linear session types together with their operational meaning.

As an illustration, we consider a protocol on how to interact with a provider of a queue data structure that contains elements of some variable type A[1]. The protocol is defined by the session type below; we will see variations of it throughout this paper.

$$\begin{aligned}\mathsf{queue}\ A = \&\{&\mathsf{enq} : A \multimap \mathsf{queue}\ A,\\ &\mathsf{deq} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : A \otimes \mathsf{queue}\ A\}\}\end{aligned}$$

The session type prescribes that a process providing a service of type $\mathsf{queue}\ A$, gives a client the choice to either enqueue (enq) or dequeue (deq) an element of type A. Upon receipt of the label enq, the providing process expects to receive a channel of type A to be enqueued and recurs. Upon receipt of the label deq, the providing process either indicates that the queue is empty (none), in which case it terminates, or that there is a channel stored in

[1]Polymorphism is orthogonal to the investigation of this paper, so we adopt it for the examples without formal treatment, which can be found in the literature [25, 47].

the queue (some), in which case it dequeues this element, sends it to the client, and recurs. We adopt an *equi-recursive* [14] interpretation for recursive session types, which requires recursive session types to be *contractive* [22]. This interpretation guarantees that there are no messages associated with the unfolding of a recursive type.

Figure 1 shows two process definitions *empty* and *elem* implementing the session type queue A. In SILL, we declare the type of a defined process X with $X : \{A \leftarrow A_1, \ldots, A_n\}$, indicating that the process provides a service of type A, using channels of type $A_1, \ldots, A_n$. The definition of the process is then given by $x \leftarrow X \leftarrow y_1, \ldots, y_n = P$ where P is a process term satisfying $y_1 : A_1, \ldots, y_n : A_n \vdash P :: (x : A)$. A new process X providing along x is spawned with an expression of the form $x \leftarrow X \leftarrow y_1, \ldots, y_n \; ; Q_x$, where Q_x is the continuation binding x. The channels $y_1, \ldots, y_n$ are passed to X and hence no longer available to Q_x.

```
elem : {queue A ← A, queue A}

q ← elem ← x, t =
  case q of
  | enq → y ← recv q ;        % x : A, t : qu A, y : A ⊢ q : qu A
          t.enq ; send t y ;  % x : A, t : qu A ⊢ q : qu A
          q ← elem ← x, t
  | deq → q.some ;            % x : A, t : qu A ⊢ q : A ⊗ qu A
          send q x ;          % t : qu A ⊢ q : qu A
          fwd q t
```

```
empty : {queue A}

q ← empty =
  case q of
  | enq → x ← recv q ;        % x : A ⊢ q : qu A
          e ← empty ;         % x : A, e : qu A ⊢ q : qu A
          q ← elem ← x, e
  | deq → q.none ;            % ⊢ q : 1
          close q
```

Figure 1: Processes implementing linear session type queue A.

The queue in Figure 1 is implemented as a sequence of *elem* processes, ending in an *empty* process. The recursive process *elem* provides a queue along channel q and uses a channel $x : A$ (the element in front of the queue) as well as a channel t : queue A (the tail of the queue). If it receives an enq label and then a channel y, it simply enqueues y in the tail t. If it receives a deq label it responds with some, followed by the channel x it holds, and then forwards all future communication along q to the tail t. The implementation is highly parallel; in particular, many enqueueing operatons can be in flight at the same time. Process *empty*, on the other hand, builds a singleton queue from an element received to be enqueued and returns none and terminates when asked to dequeue. Perhaps the most unusual aspect of writing session-typed programs is that the type of channels changes during interactions, as already indicated in Table 1. To make this explicit we annotate the code in Figure 1 with the types of all channels at the various points in a process definition. We abbreviate queue A to qu A in those comments.

3 Manifest Sharing

In this section, we extend the linear process language of the previous section with the capability to share a process among several clients. The shared channels introduced previously [8, 62] via the exponential modality in linear logic have a copying semantics and therefore do not allow sharing of mutable resources as pursued in this paper. We first approach the support of shared processes programmatically, by introducing acquire-release as a primitive to our language. We then derive those primitives as modalities from logic in a stratified system of session types. Lastly, we develop the notion of an equi-synchronizing session type.

3.1 A Programming Perspective

In the intuitionistic linear setting of Section 2, processes form a tree at run-time, guaranteeing that a client of a process is the only client of that process. With the introduction of *shared processes* this invariant no longer holds because there may exist multiple clients that refer to the process by a *shared channel*. To uphold session fidelity, communication along a shared channel must only be possible once exclusive access to the process providing along that channel has been obtained. To this end, we impose an *acquire-release* discipline on shared processes, where an acquire yields exclusive access to a shared process, if the process is available, and a release relinquishes exclusive access. As a result, processes can alternate between linear and shared, where a successful acquire of a shared process turns the process into a linear one, and conversely, a release of a linear process turns the process into a shared one. This view of a process undergoing phases requires an identification of a process with a thread of control, which is extremely natural in intuitionistic linear logic since we can identify a process with the channel along which it provides a service.

We illustrate the programmatic working of the acquire-release primitives on a schematic producer-consumer scenario in Figure 2. For now, we assume for both processes that the shared channel q is provided by a shared process of session type $\mathsf{queue}\ A$ that stores shared elements x of type A. In program code, we typeset shared channels as well as shared session types in red and **bold** font to make them distinguishable from linear channels and session types, which we typeset in black and regular font. Moreover, we assume that the session type $\mathsf{queue}\ A$ recurs rather than terminates upon dequeueing, if the queue is empty, which is more appropriate for a producer-consumer context. In Section 3.2 we clarify how to change the type specification to accommodate these assumptions.

Processes *produce* and *consume* in Figure 2 attempt to communicate with the queue by issuing corresponding acquire and release statements. Process *produce*, for example, issues the statement $q' \leftarrow \mathsf{acquire}\ q$, which, if successful, yields the queue's linear channel q' along which the process can enqueue the element. Before the process recurs, it releases the now linear queue process providing along q' by issuing $q \leftarrow \mathsf{release}\ q'$. This yields the queue's shared channel q and gives turn to another producer or consumer.

```
produce : {1 ← A, queue A}              consume : {1 ← queue A}

c ← produce ← x, q =                    c ← consume ← q =
   q' ← acquire q ;                        q' ← acquire q ;
   q'.enq ;                                q'.deq ;
   send q' x ;                             case q' of
   q ← release q' ;                        | some → x ← recv q' ;
   c ← produce ← x, q                                q ← release q' ;
                                                     c ← consume ← q
                                           | none → q ← release q' ;
                                                     c ← consume ← q
```

Figure 2: Acquire-release primitives illustrated on producer-consumer, programmatically. Shared channels are typeset in red and **bold** font, linear channels in black and regular font. See Section 3.2 for definition of shared session type $\mathsf{queue}\ A$.

3.2 A Logic Perspective

Like send and receipt of a message, acquire and release denote synchronization points in the communication between processes. If we were to introduce acquire and release as operational primitives only, session types would no longer accurately prescribe the protocols of communication. To restore the descriptive power of session types, we enrich the type system so that the type of a process dictates at which points in the communication acquire and release must happen.

The key idea in pursuit of this goal is to generalize the notion of type *stratification* introduced in [49], based on Benton's LNL 1994 and Reed's adjoint logic 2009, and to stratify session types into a *linear* and *shared* layer. We then connect these layers with *modalities* that go back and forth between them. In Pfenning and Griffith 2015 the modes are U, F, and L for unrestricted, affine, and linear session types, respectively. In this paper, we focus on the interplay between the modes S and L, pertaining to shared and linear session types, respectively. An integration with the remaining modes U and F is straightforward.

The stratification arises from a difference in structural properties that exist for session types at a mode — or propositions, when viewed through the lens of the Curry-Howard correspondence. For example, shared propositions can be weakened, contracted, and exchanged, whereas linear propositions can only be exchanged. The difference in structural properties entails a hierarchy between modes such that a mode with fewer structural properties is at the bottom. The hierarchy for the modes S and L is:

$$\mathsf{S} > \mathsf{L}$$

The *independence principle* for modes states that proofs of a proposition of a stronger mode (with more structural properties) may not depend on hypotheses of a strictly weaker mode (with fewer structural properties). This is because a client of a stronger proposition may, for example, reuse the proposition, which would implicitly reuse the weaker proposition on which it depends. More technically, on the logical side, cut elimination would fail without this restriction. As a result, we get separate[2] hypothetical judgments for shared and linear processes which, by definition, obey the independence principle:

$$\Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

The subscripts denote the respective mode of a channel or session type, and the contexts Γ and Δ consist of hypotheses on the typing of shared and linear channels, respectively. The judgments depend on a signature Σ that is populated with all process definitions prior to type-checking, allowing for recursive process definitions.

Given the two layers, we can now define the modality $\downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}}$, which shifts a shared proposition (session type) to a linear one, and the modality $\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$, which shifts a linear proposition (session type) to a shared one. The resulting strata restricted to session types (propositions) at the modes S and L are:[3]

$$\begin{array}{lcl} A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\ A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid \exists A_{\mathsf{S}}.B_{\mathsf{L}} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \Pi A_{\mathsf{S}}.B_{\mathsf{L}} \mid \&\{\overline{l : A_{\mathsf{L}}}\} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \end{array}$$

We review the new connectives and their operational meaning in Table 2. Together with Table 1, this table defines the connectives supported in $\mathsf{SILL}_{\mathsf{S}}$. Besides the new connectives to accommodate acquire-release, which we discuss in more detail below, we introduce the connectives $\Pi A_{\mathsf{S}}.B_{\mathsf{L}}$ and $\exists A_{\mathsf{S}}.B_{\mathsf{L}}$ to support shared channel input and output, respectively. These connectives of mixed mode are based on the dependent session type connectives introduced in [12, 64], yet at the present stage our language does not make use of the potentially dependent nature of these connectives. The process typing rules for the connectives of $\mathsf{SILL}_{\mathsf{S}}$, excluding the acquire-release connectives, which we discuss below, can be found in Figure 3. A complete listing of all the process typing rules can be found in Figure 17 and Figure 18 in the appendix.

Session type		**Process term**		
current	**continuation**	**current**	**continuation**	**Description**
$c_{\mathsf{L}} : \Pi A_{\mathsf{S}}.B_{\mathsf{L}}$	$c_{\mathsf{L}} : B_{\mathsf{L}}$	$\mathsf{send}\ c_{\mathsf{L}}\ d_{\mathsf{S}}\ ; P$	P	provider sends channel $d_{\mathsf{S}} : A_{\mathsf{S}}$ along c_{L}
		$y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; Q_{y_{\mathsf{S}}}$	$[d_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}}$	client receives channel $d_{\mathsf{S}} : A_{\mathsf{S}}$ along c_{L}
$c_{\mathsf{L}} : \exists A_{\mathsf{S}}.B_{\mathsf{L}}$	$c_{\mathsf{L}} : B_{\mathsf{L}}$	$y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; P_{y_{\mathsf{S}}}$	$[d_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{y_{\mathsf{S}}}$	provider receives channel $d_{\mathsf{S}} : A_{\mathsf{S}}$ along c_{L}
		$\mathsf{send}\ c_{\mathsf{L}}\ d_{\mathsf{S}}\ ; Q$	Q	client sends channel $d_{\mathsf{S}} : A_{\mathsf{S}}$ along c_{L}
$c_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}}$	$c_{\mathsf{S}} : A_{\mathsf{S}}$	$c_{\mathsf{S}} \leftarrow \mathsf{detach}\ c_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}}$	$[c_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}$	provider sends "$\mathsf{detach}\, c_{\mathsf{S}}$" along c_{L}
		$x_{\mathsf{S}} \leftarrow \mathsf{release}\ c_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}}$	$[c_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}$	client receives "$\mathsf{detach}\, c_{\mathsf{S}}$" along c_{L}
$c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$	$c_{\mathsf{L}} : A_{\mathsf{L}}$	$c_{\mathsf{L}} \leftarrow \mathsf{acquire}\ c_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}}$	$[c_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}$	client sends "$\mathsf{acquire}\, c_{\mathsf{L}}$" along c_{S}
		$x_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}}$	$[c_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}$	provider receives "$\mathsf{acquire}\, c_{\mathsf{L}}$" along c_{S}

Table 2: Overview of shared session types together with their operational meaning. See Table 1 for linear connectives.

We are now in a position to define the typing of the acquire-release discipline outlined in the previous section. In particular, we must determine what the types of the channels should be to which acquire and release are applied. Observing that an acquire transforms a shared channel into a linear one, the natural choice is to type the shared channel of an acquire with the modality $\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$. Analogously, the linear channel of a release should be typed with the modality $\downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}}$ as it transforms a linear channel into a shared one. Because we adopt an intuitionistic formulation, which avoids the need for explicit dualization of a session type, we get both a left and right rule for each primitive. The notions of acquire and release are naturally formulated from the point of view of a client, so we use those terms in the left rules. For the right rules, we use the terms *accept* and *detach* with the meaning that an accept accepts

[2]We could have chosen an combined judgment with a combined context and corresponding projections onto each mode, as employed in [49] for a richer structure of modes. For this paper, we have chosen separate judgments and contexts for clarity of presentation.

[3]Shared counterparts of all the linear connectives can be defined at the shared level as well, but for the purposes of this paper we will keep the shared layer as simple as possible.

$$\frac{}{\Gamma;\ y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{L}}\ y_{\mathsf{L}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}\ (\text{T-Id}_{\mathsf{L}}) \qquad \frac{\hat{A} \leq A_{\mathsf{S}}}{\Gamma, y_{\mathsf{S}} : \hat{A} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{S}}\ y_{\mathsf{S}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}\ (\text{T-Id}_{\mathsf{S}})$$

$$\frac{\Gamma = \overline{w_{\mathsf{S}} : \hat{B}} \quad \overline{\hat{B}} \leq \overline{B_{\mathsf{S}}} \quad \Delta = \overline{y_{\mathsf{L}} : B_{\mathsf{L}}} \quad (x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}' : B_{\mathsf{L}}}, \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}'}, \overline{w_{\mathsf{S}}'}}) \in \Sigma \quad \Gamma, \Gamma';\ \Delta', x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, \Gamma';\ \Delta, \Delta' \vdash_\Sigma x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-Spawn}_{\mathsf{LL}})$$

$$\frac{\Gamma = \overline{y_{\mathsf{S}} : \hat{B}} \qquad \overline{\hat{B}} \leq \overline{B} \qquad (x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}' : B} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}'}}) \in \Sigma \qquad \Gamma, \Gamma', x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, \Gamma';\ \Delta \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-Spawn}_{\mathsf{LS}})$$

$$\frac{\Gamma = \overline{y_{\mathsf{S}} : \hat{B}} \qquad \overline{\hat{B}} \leq \overline{B} \qquad (x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}' : B} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}'}}) \in \Sigma \qquad \Gamma, \Gamma', x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (z_{\mathsf{S}} : C_{\mathsf{S}})}{\Gamma, \Gamma' \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{S}}} :: (z_{\mathsf{S}} : C_{\mathsf{S}})}\ (\text{T-Spawn}_{\mathsf{SS}})$$

$$\frac{\Gamma;\ \Delta \vdash_\Sigma Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \mathbf{1} \vdash_\Sigma \mathsf{wait}\ x_{\mathsf{L}}\ ; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\mathbf{1}_{\mathrm{L}}) \qquad \frac{}{\Gamma;\ \cdot \vdash_\Sigma \mathsf{close}\ x_{\mathsf{L}} :: (x_{\mathsf{L}} : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\mathrm{R}})$$

$$\frac{\Gamma;\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}} \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\ x_{\mathsf{L}}\ ; Q_{y_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\otimes_{\mathrm{L}}) \qquad \frac{\Gamma;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\ x_{\mathsf{L}}\ y_{\mathsf{L}}\ ; P :: (x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})}\ (\text{T-}\otimes_{\mathrm{R}})$$

$$\frac{\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\ x_{\mathsf{L}}\ ; Q_{y_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\exists_{\mathrm{L}}) \qquad \frac{\hat{A} \leq A_{\mathsf{S}} \qquad \Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta \vdash_\Sigma \mathsf{send}\ x_{\mathsf{L}}\ y_{\mathsf{S}}\ ; P :: (x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}))}\ (\text{T-}\exists_{\mathrm{R}})$$

$$\frac{\Gamma;\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\ x_{\mathsf{L}}\ y_{\mathsf{L}}\ ; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\multimap_{\mathrm{L}}) \qquad \frac{\Gamma;\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\ x_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}})}\ (\text{T-}\multimap_{\mathrm{R}})$$

$$\frac{\hat{A} \leq A_{\mathsf{S}} \qquad \Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta, x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\ x_{\mathsf{L}}\ y_{\mathsf{S}}\ ; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\Pi_{\mathrm{L}}) \qquad \frac{\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\ x_{\mathsf{L}}\ ; P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}))}\ (\text{T-}\Pi_{\mathrm{R}})$$

$$\frac{(\forall i)\ \ \Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}_i} \vdash_\Sigma Q_i :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\} \vdash_\Sigma \mathsf{case}\ x_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\oplus_{\mathrm{L}}) \qquad \frac{\Gamma;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}\,h})}{\Gamma;\ \Delta \vdash_\Sigma x_{\mathsf{L}}.l_h\ ; P :: (x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})}\ (\text{T-}\oplus_{\mathrm{R}})$$

$$\frac{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}\,h} \vdash_\Sigma Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\} \vdash_\Sigma x_{\mathsf{L}}.l_h\ ; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\&_{\mathrm{L}}) \qquad \frac{(\forall i)\ \ \Gamma;\ \Delta \vdash_\Sigma P_i :: (x_{\mathsf{L}} : A_{\mathsf{L}_i})}{\Gamma;\ \Delta \vdash_\Sigma \mathsf{case}\ x_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P} :: (x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})}\ (\text{T-}\&_{\mathrm{R}})$$

Figure 3: Remaining process typing rules not shown inline.

an acquire and a detach initiates a release. We review each pair of rules in turn, along with their operational semantics:

The typing of the pair acquire-accept is defined by the following rules:

$$\frac{\Gamma, x_{\mathsf{S}} : {\uparrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : {\uparrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}{\uparrow}_{\mathrm{L}}^{\mathsf{S}}\mathrm{L}) \qquad \frac{\Gamma;\ \cdot \vdash_\Sigma P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : {\uparrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\text{T-}{\uparrow}_{\mathrm{L}}^{\mathsf{S}}\mathrm{R})$$

An acquire is applied to the shared channel x_{S} along which the shared process offers and yields the process' linear channel x_{L}, when successful. The shared channel x_{S} is still available to the continuation $Q_{x_{\mathsf{L}}}$. By accepting an acquire request by a client along its shared channel x_{S}, a shared process transitions to a linear process, now offering along its linear channel x_{L}. Since the independence principle forbids a shared process to depend on linear channels, the now linear process starts out with an empty linear context.

(D-$\text{I}\text{D}_{\mathsf{L}}$)	$\mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}) \longrightarrow a_{\mathsf{L}} = b_{\mathsf{L}},\, a_{\mathsf{S}} = b_{\mathsf{S}}$
(D-$\text{I}\text{D}_{\mathsf{S}}$)	$\mathsf{proc}(a_{\mathsf{S}},\, \mathsf{fwd}\ a_{\mathsf{S}}\ b_{\mathsf{S}}) \longrightarrow \mathsf{unavail}(a_{\mathsf{S}}),\, a_{\mathsf{S}} = b_{\mathsf{S}}$
(D-$\text{S}\text{PAWN}_{\mathsf{LL}}$)	$\mathsf{proc}(a_{\mathsf{L}},\, x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{c_{\mathsf{L}}}, \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{L}}}) \longrightarrow \mathsf{proc}(a_{\mathsf{L}},\, [b_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\, \mathsf{proc}(b_{\mathsf{L}},\, [b_{\mathsf{L}}/x'_{\mathsf{L}},\, \overline{c_{\mathsf{L}}/y_{\mathsf{L}}},\, \overline{c_{\mathsf{S}}/y_{\mathsf{S}}}]P_{x'_{\mathsf{L}},\overline{y_{\mathsf{L}}},\overline{y_{\mathsf{S}}}}),\, \mathsf{unavail}(b_{\mathsf{S}})$ for $x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}, \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}},\overline{y_{\mathsf{L}}},\overline{y_{\mathsf{S}}}} \in \Sigma$ and b *fresh*
(D-$\text{S}\text{PAWN}_{\mathsf{LS}}$)	$\mathsf{proc}(a_{\mathsf{L}},\, x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}) \longrightarrow \mathsf{proc}(a_{\mathsf{L}},\, [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\, \mathsf{proc}(b_{\mathsf{S}},\, [b_{\mathsf{S}}/x'_{\mathsf{S}},\, \overline{c_{\mathsf{S}}/y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}})$ for $x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} \in \Sigma$ and b *fresh*
(D-$\text{S}\text{PAWN}_{\mathsf{SS}}$)	$\mathsf{proc}(a_{\mathsf{S}},\, x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}) \longrightarrow \mathsf{proc}(a_{\mathsf{S}},\, [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\, \mathsf{proc}(b_{\mathsf{S}},\, [b_{\mathsf{S}}/x'_{\mathsf{S}},\, \overline{c_{\mathsf{S}}/y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}})$ for $x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} \in \Sigma$ and b *fresh*
(D-$\mathbf{1}$)	$\mathsf{proc}(c_{\mathsf{L}},\, \mathsf{wait}\ a_{\mathsf{L}}\,;\, Q),\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{close}\ a_{\mathsf{L}}) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, Q)$
(D-$\otimes/\exists$)	$\mathsf{proc}(c_{\mathsf{L}},\, y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\,;\, Q_y),\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{send}\ a_{\mathsf{L}}\ b\,;\, P) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, [b/y]\, Q_y),\, \mathsf{proc}(a_{\mathsf{L}},\, P)$
(D-$\multimap/\Pi$)	$\mathsf{proc}(c_{\mathsf{L}},\, \mathsf{send}\ a_{\mathsf{L}}\ b\,;\, Q),\, \mathsf{proc}(a_{\mathsf{L}},\, y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\,;\, P_y) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, Q),\, \mathsf{proc}(a_{\mathsf{L}},\, [b/y]P_y)$
(D-$\oplus$)	$\mathsf{proc}(c_{\mathsf{L}},\, \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\, \mathsf{proc}(a_{\mathsf{L}},\, a_{\mathsf{L}}.l_h\,;\, P) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, Q_h),\, \mathsf{proc}(a_{\mathsf{L}},\, P)$
(D-$\&$)	$\mathsf{proc}(c_{\mathsf{L}},\, a_{\mathsf{L}}.l_h\,;\, Q),\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}) \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, Q),\, \mathsf{proc}(a_{\mathsf{L}},\, P_h)$

Figure 4: Remaining multiset rewriting rules not shown inline.

Operationally, we capture the dynamics of $\mathsf{SILL_S}$ by *multiset rewriting rules* [11]. A multiset rewriting rule is generally of the form $S_1, \ldots, S_n \longrightarrow T_1, \ldots, T_m$ and denotes a transition from $S_1, \ldots, S_n$ to the $T_1, \ldots, T_m$ where each S_i and T_j is a formula capturing some aspect of the current state of the computation. In our setting, we use the rules to capture a transition in the configuration of processes that arise from a program. As we discuss in Section 5.1, we use the predicates $\mathsf{proc}(c_m,\, P)$ and $\mathsf{unavail}(a_{\mathsf{S}})$ to define the states of a configuration. The former denotes a process with process term P that provides along channel c_m at mode m, the latter acts as a placeholder for a shared process providing along channel a_{S} that is currently not available. Multiset rewriting rules are local in that they only mention the parts of a configuration they rewrite. The synchronous dynamics of the pair acquire-accept is given by the following rule:

$$\begin{array}{l} \mathsf{proc}(c_{\mathsf{L}},\, x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\,;\, Q_{x_{\mathsf{L}}}),\, \mathsf{proc}(a_{\mathsf{S}},\, x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\,;\, P_{x_{\mathsf{L}}}) \\ \quad \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\, \mathsf{proc}(a_{\mathsf{L}},\, [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\, \mathsf{unavail}(a_{\mathsf{S}}) \end{array} \qquad (\text{D-}\uparrow^{\mathsf{S}}_{\mathsf{L}})$$

The above rule indicates that, at run-time, for every process there exists one channel, a, with two modes, a linear one, a_{L}, and a shared one, a_{S}. When a process shifts between modes, it switches between the two modes of its offering channel. This channel is substituted at the appropriate mode for the variables occurring within process terms. The typing rules (T-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$L) and (T-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$R) create a fresh variable x_{L}, for which the already existing channel a at mode L is substituted. The offering channel together with its two modes is created when a process is spawned.

Figure 4 gives the dynamics of the remaining connectives in $\mathsf{SILL_S}$. A complete listing of all the multiset rewriting rules can be found in Figure 20 in the appendix. The side condition b *fresh* indicates allocation of a globally fresh channel and the equality $a = b$ expresses that b is substituted for a in the entire configuration. For convenience, we write multiset rewriting rules such that a providing process appears to the right of its client. Multiset rewriting rules, however, are unordered.

The typing of the pair release-detach is defined by the following rules:

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\,;\, Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\text{L}) \qquad \frac{\Gamma \vdash_{\Sigma} P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\,;\, P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}})}\ (\text{T-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\text{R})$$

The rules are essentially inverse to the typing rules of acquire-release; we point out that rule (T-${\downarrow}^{\mathsf{S}}_{\mathsf{L}}$R) requires the linear context to be empty, to satisfy the independence principle. Operationally, the rules have the following semantics:

$$\begin{array}{l} \mathsf{proc}(c_{\mathsf{L}},\, x_{\mathsf{S}} \leftarrow \mathsf{release}\ a_{\mathsf{L}}\,;\, Q_{x_{\mathsf{S}}}),\, \mathsf{proc}(a_{\mathsf{L}},\, x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\,;\, P_{x_{\mathsf{S}}}),\, \mathsf{unavail}(a_{\mathsf{S}}) \\ \quad \longrightarrow \mathsf{proc}(c_{\mathsf{L}},\, [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\, \mathsf{proc}(a_{\mathsf{S}},\, [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}) \end{array} \qquad (\text{D-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}})$$

This time the rules shift the process from S to L, by switching the offering channel from a_{L} to a_{S} and by substituting the channel a_{S} for the variable x_{S}.

Let's now return to the producer-consumer example and work out what the type specifications have to be. The processes *produce* and *consume* in Figure 2 have been devised under the assumption that the channel q is a shared channel to a shared queue and that the shared queue process recurs rather than terminates upon dequeueing, if the queue is empty. For this to be the case, we change the session type queue from Section 2 as follows:

$$\begin{aligned}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\&\{&\mathsf{enq} : \Pi \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\ \mathsf{some} : \exists \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}}\}\}\end{aligned}$$

With this change, the code in Figure 2 is type-correct as it is written. The new definition of session type $\mathsf{queue}\ A_{\mathsf{S}}$ uses the previously introduced dependent linear session types $\Pi A_{\mathsf{S}}.B_{\mathsf{L}}$ and $\exists A_{\mathsf{S}}.B_{\mathsf{L}}$ for shared channel input and output, respectively, and prescribes the following synchronization pattern: When a process of type $\mathsf{queue}\ A_{\mathsf{S}}$ is spawned, it starts out as a shared process that first must be acquired. Any of the defined sequences of inputs and outputs then are executed while the process is linear. After such an exchange, the process recurs at type ${\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathsf{queue}\ A_{\mathsf{S}}$. Since $\mathsf{queue}\ A_{\mathsf{S}}$ is defined as ${\uparrow}^{\mathsf{S}}_{\mathsf{L}}\&\{\dots\}$, the type ${\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{queue}\ A_{\mathsf{S}}$ amounts to the type ${\downarrow}^{\mathsf{S}}_{\mathsf{L}}{\uparrow}^{\mathsf{S}}_{\mathsf{L}}\&\{\dots\}$. This means that in its recursion, the process will first need to be released to become a shared process of type $\mathsf{queue}\ A_{\mathsf{S}}$. Looking at the implementations of processes *produce* and *consume* in Figure 2, we can see that they comply with the acquire-release pattern dictated by the above session type. For example, after process *produce* has sent the channel x along channel q', the channel q' is of type ${\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathsf{queue}\ A_{\mathsf{S}}$, which is why process *produce* releases that channel before it recurs.

Having changed the specification of session type $\mathsf{queue}\ A_{\mathsf{S}}$, we must correspondingly change the implementations of processes *empty* and *elem* shown in Figure 1; the result is given in Figure 5. The code predominantly contains the matching pairs accept and detach as well as acquire and release, respectively. For example, the first statement in process *empty* accepts an acquire request from a client. Similarly, the statement $q \leftarrow \mathsf{detach}\ q'$ initiates a release by a client.

```
empty : {queue A_S}                      elem : {queue A_S ← A_S, queue A_S}

q ← empty =                              q ← elem ← x, t =
  q′ ← accept q ;                          q′ ← accept q ;
  case q′ of                               case q′ of
  | enq → x ← recv q′ ;                    | enq → y ← recv q′ ;
          e ← empty ;                              t′ ← acquire t ;
          q ← detach q′ ;                          t′.enq ; send t′ y ;
          q ← elem ← x, e                          t ← release t′ ;
  | deq → q′.none ;                                q ← detach q′ ;
          q ← detach q′ ;                          q ← elem ← x, t
          q ← empty                        | deq → q′.some ;
                                                   send q′ x ;
                                                   q ← detach q′ ;
                                                   fwd q t
```

Figure 5: Implementation of a shared queue. See Figure 1 for linear version.

Session type $\mathsf{queue}\ A_{\mathsf{S}}$ pinpoints a typical pattern of shared process programming where a shared recursive session type $Y_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}A_{\mathsf{L}}$ recurs at type ${\downarrow}^{\mathsf{S}}_{\mathsf{L}}Y_{\mathsf{S}}$. The benefits of this pattern are two-fold: on the one hand, it guarantees that the session type Y_{S} allows for perpetual acquire-release cycles and, on the other hand, it makes sure that all acquired processes are released at recursion point because linearity forbids any linear channels to be left behind.

Comparing this shared version of session type queue with its linear version in Section 2, we note that the shared version does no longer require the elements stored in the queue to be linear. The elements can either be of a shared type A_{S} or of a shifted linear type ${\uparrow}^{\mathsf{S}}_{\mathsf{L}}A_{\mathsf{L}}$. This choice is not available for a linear queue because the queue can only be consumed if all the elements stored in it can be consumed too. For the same reason it is now also possible for the shared queue to recur upon dequeueing rather than terminating, if the queue is empty.

3.3 Equi-Synchronizing Session Types

So far we have achieved that a client communicates with a shared process in mutual exclusion from other clients and that the acquire and release points of a shared process manifest in its session type. There remains a last threat to session fidelity that we need to address: erroneous assumptions by a client on a shared process' type. These can come about, for example, in the following scenario: two clients Q_1 and Q_2 are trying to acquire access to the same shared channel c_{S} at type $\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$. Let's assume that Q_1 succeeds and then later releases c_{L} to a *different* type $\uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$. Once Q_2 finally obtains access to c_{S}, Q_2 and the provider will disagree on the type of the channel c_{L}: the provider will think that $c_{\mathsf{L}} : B_{\mathsf{L}}$, while Q_2 will think that $c_{\mathsf{L}} : A_{\mathsf{L}}$, thereby violating session fidelity.

To guarantee preservation without resorting to run-time checks, we introduce the notion of an *equi-synchronizing* session type. A session type is equi-synchronizing if it imposes the invariant on a process to be *released* to the *same* type at which the process was previously *acquired*. No constraint is imposed on channels that were never acquired. For example, our shared $\mathsf{queue}\ A_{\mathsf{S}}$ from Section 3.2

$$\begin{aligned}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \& \{ & \mathsf{enq} : \Pi \mathbf{A}_{\mathsf{S}}. \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{queue}\ \mathbf{A}_{\mathsf{S}}, \\ & \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\ \mathsf{some} : \exists \mathbf{A}_{\mathsf{S}}. \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{queue}\ \mathbf{A}_{\mathsf{S}}\}\}\end{aligned}$$

is equi-synchronizing because, in each branch, it releases a channel back to type $\mathsf{queue}\ A_{\mathsf{S}}$, which is the type at which the channel must have been acquired.

We formally define the notion of an equi-synchronizing session type in Figure 6, giving a coinductive definition. The definition is based on the judgment

$$\vdash_{\Sigma} (A, \hat{D})\ \mathsf{esync}$$

where $\hat{D}$ represents a constraint on the type at which a channel of type A can be released to. If $\hat{D} = \top$, then there is no constraint on a future release, if $\hat{D} = D_{\mathsf{S}}$ then any release must take place to type D_{S}. There is a third possibility, $\hat{D} = \bot$ which means that A may never be released. This is necessary only for the proof of session fidelity (see Section 5.2).

We say that the type A *equi-synchronizes* to $\hat{D}$ or that $\hat{D}$ is A's equi-synchronizing constraint. Underlying the coinductive definition of an equi-synchronizing session type is the notion of a *continuation type*. To check that a type A equi-synchronizes to the type $\uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}}$, the rules in Figure 6 transitively step through A's continuation (starting from $(A, \top)$) until the first acquisition point $\uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$ is encountered. At this point, the type $\uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$ is set to be the equi-synchronizing constraint, and the rules transitively step through each continuation of B_{L} until the first release point $\downarrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{S}}$ is encountered. The session type is equi-synchronizing, if it holds that $C_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}}$ at each such release point.

$$\frac{(\forall i)\ \vdash_{\Sigma} (A_{\mathsf{L}_i}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (\oplus\{\overline{l : A_{\mathsf{L}}}\}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\oplus}) \qquad \frac{(\forall i)\ \vdash_{\Sigma} (A_{\mathsf{L}_i}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (\&\{\overline{l : A_{\mathsf{L}}}\}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\&})$$

$$\frac{\vdash_{\Sigma} (B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (A_{\mathsf{L}} \otimes B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\otimes}) \qquad \frac{\vdash_{\Sigma} (B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (A_{\mathsf{L}} \multimap B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\multimap})$$

$$\frac{\vdash_{\Sigma} (B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (\exists A_{\mathsf{S}}.B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\exists}) \qquad \frac{\vdash_{\Sigma} (B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}{\vdash_{\Sigma} (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\Pi}) \qquad \frac{}{\vdash_{\Sigma} (\mathbf{1}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_{\mathbf{1}})$$

$$\frac{\vdash_{\Sigma} (A_{\mathsf{L}}, \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})\ \mathsf{esync}}{\vdash_{\Sigma} (\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}, \top)\ \mathsf{esync}}\ (\text{T-Esync}_{\uparrow_{\mathsf{L}}^{\mathsf{S}}}) \qquad \frac{\vdash_{\Sigma} (D_{\mathsf{S}}, \top)\ \mathsf{esync}}{\vdash_{\Sigma} (\downarrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{S}}, D_{\mathsf{S}})\ \mathsf{esync}}\ (\text{T-Esync}_{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\text{-1}) \qquad \frac{\vdash_{\Sigma} (D_{\mathsf{S}}, \top)\ \mathsf{esync}}{\vdash_{\Sigma} (\downarrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{S}}, \top)\ \mathsf{esync}}\ (\text{T-Esync}_{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\text{-2})$$

Figure 6: Equi-synchronizing session type, coinductively defined.

Let's exercise the rules in Figure 6 on our shared $\mathsf{queue}\ A_{\mathsf{S}}$. We start out with $\vdash_{\Sigma} (\mathsf{queue}\ A_{\mathsf{S}},\ \top)\ \mathsf{esync}$. Since session types are interpreted equi-recursively and are contractive [22], we can "silently" replace $\mathsf{queue}\ A_{\mathsf{S}}$ with its definition, which means we have to check

$$\vdash_{\Sigma} (\uparrow_{\mathsf{L}}^{\mathsf{S}} \& \{\mathsf{enq} : \ldots\ \mathsf{deq} : \ldots\},\ \top)\ \mathsf{esync}$$

According to rule (T-ESYNC$_{\uparrow^{\mathsf{S}}_{\mathsf{L}}}$), we set the equi-synchronizing constraint to $\mathsf{queue}\ A_{\mathsf{S}}$, requiring us to check for the continuation that

$$\vdash_\Sigma (\&\{\mathsf{enq} : \ldots\ \mathsf{deq} : \ldots\},\ \mathsf{queue}\ A_{\mathsf{S}})\ \ \mathsf{esync}$$

According to rule (T-ESYNC$_{\&}$), we are required to check for each continuation that

$$\vdash_\Sigma (\Pi A_{\mathsf{S}}.{\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{queue}\ A_{\mathsf{S}})\ \ \mathsf{esync}$$

$$\vdash_\Sigma (\oplus\{\mathsf{none} : {\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists A_{\mathsf{S}}.{\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathsf{queue}\ A_{\mathsf{S}}\},\ \mathsf{queue}\ A_{\mathsf{S}})\ \ \mathsf{esync}$$

Let's consider the first branch. According to rule (T-ESYNC$_\Pi$) we must check that

$$\vdash_\Sigma ({\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{queue}\ A_{\mathsf{S}})\ \ \mathsf{esync}$$

which, according to rule (T-ESYNC$_{\downarrow^{\mathsf{S}}_{\mathsf{L}}}$-1), amounts to the check

$$\vdash_\Sigma (\mathsf{queue}\ A_{\mathsf{S}},\ \top)\ \mathsf{esync}$$

This is the check we started out with, allowing us to succeed on this branch since our rules are interpreted coinductively. Since the same holds true for all branches, we conclude that the session type $\mathsf{queue}\ A_{\mathsf{S}}$ is equi-synchronizing, unfolding type definitions where necessary.

Not all branches must actually release. For example, the variant

$$\begin{aligned}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}} = {\uparrow^{\mathsf{S}}_{\mathsf{L}}}\&\{&\mathsf{enq} : \Pi \mathbf{A}_{\mathsf{S}}.{\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \exists \mathbf{A}_{\mathsf{S}}.{\downarrow^{\mathsf{S}}_{\mathsf{L}}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}}\}\}\end{aligned}$$

of the shared queue above is equi-synchronizing even though the queue terminates upon dequeuing in case of an empty queue. In that case, the queue can effectively no longer be acquired.

As we will show in more detail in Section 5.2, the equi-synchronizing invariants are at the core of the preservation proof, requiring us to show that each process maintains its equi-synchronizing constraint along all possible transitions. The three possible constraints $\hat{D}$, namely $\top$, ${\uparrow^{\mathsf{S}}_{\mathsf{L}}}A_{\mathsf{L}}$, and $\perp$, are related by the following partial order, for any A_{L}:

$$\top \geq {\uparrow^{\mathsf{S}}_{\mathsf{L}}}A_{\mathsf{L}} \geq \perp$$

This relationship becomes relevant for substitutions, where we allow substituting a channel of a smaller type for variables or channels of a bigger type at the client side (see Section 5.2).

When checking the signature Σ, recursive session type definitions are checked to be both contractive and equi-synchronizing and process definitions are checked to provide an equi-synchronizing session type. The check is initiated with $\top$ as a constraint to convey that any initial release is unconstrained. A purely linear session type A_{L} with neither acquire nor release points will thus satisfy the constraint $\vdash_\Sigma (A_{\mathsf{L}}, \top)\ \mathsf{esync}$ and also the even stronger condition $\vdash_\Sigma (A_{\mathsf{L}}, \perp)\ \ \mathsf{esync}$.

4 More Examples

In this section, we illustrate manifest sharing on several examples. Section E provides additional examples, including an "imperative" style of a queue implementation that maintains a reference to the back of the queue.

4.1 Dining Philosophers

The dining philosophers problem [19] is a prime example designed to illustrate the issues of enforcing mutual exclusion in the presences of circular dependencies among processes. It's precisely because of circularity that the dining philosophers problem cannot be modelled in the purely linear language presented in Section 2. With sharing at our disposal, however, we are now able to model the dining philosophers problem. The result is given in Figure 7.

$\mathsf{lfork} = {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathbf{sfork}$

$\mathbf{sfork} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{lfork}$

$\mathsf{phil} = \mathbf{1}$

```
fork_proc : {sfork}
c ← fork_proc =
  c′ ← accept c ;
  c ← detach c′ ;
  c ← fork_proc
```

```
thinking : {phil ← sfork, sfork}
c ← thinking ← left, right =
  (* thinking *)
  left′ ← acquire left ;
  right′ ← acquire right ;
  c ← eating ← left′, right′
```

```
eating : {phil ← lfork, lfork}
c ← eating ← left′, right′ =
  (* eating *)
  right ← release right′ ;
  left ← release left′ ;
  c ← thinking ← left, right
```

Figure 7: Dining philosophers.

The implementation defines the mutually dependent session types lfork and sfork and the session type phil, representing a fork and a philosopher, respectively. In support of the spirit of the example, the former allow perpetual acquire-release cycles and are implemented by process *fork_proc*. Session type phil, on the other hand, denotes a trivial linear session, which is implemented by the processes *thinking* and *eating*. As the names suggest, process *thinking* represents a philosopher that is thinking, whereas process *eating* represents a philosopher that is eating. A thinking philosopher has shared channel references to the forks on their left and right. Once the philosopher is done thinking, they attempt to acquire their right and left fork and transition to eating, if successful. An eating philosopher, on the other hand, has linear channel references to the forks on their left and right, which they release, once they are done eating and before transitioning to thinking. We can set up a table of 4 philosophers using the following lines of code:

```
f0 ← fork_proc ; f1 ← fork_proc ; f2 ← fork_proc ; f3 ← fork_proc ;
p0 ← thinking ← f0, f1 ; p1 ← thinking ← f1, f2 ; p2 ← thinking ← f2, f3 ; p3 ← thinking ← f3, f0 ;
```

The above setup faithfully matches the circular table and can lead to a deadlock, as pointed out by Dijkstra, if every philosopher picks up the fork on their left and then blocks, waiting for the fork on their right. We can avoid this deadlock by following Dijkstra's originally proposed solution to impose a partial order on the forks and acquiring the forks in ascending order. This can be achieved by reversing the order of the arguments in the last line to *p3* ← *thinking* ← *f0*, *f3*.

4.2 Atomicity

Another benefit of making the acquire and release points of a process manifest in the type structure is that *atomic* sections [21] become explicit. Since the statements between an up- and a downshift are executed while the process is linear, they are guaranteed to be executed without interference.

We illustrate atomicity on the example of printing to standard out from a concurrent program. To make sure that the print statements will be issued to standard out in the order that they appear in a given thread, we represent the standard output stream by a shared process that obeys the mutually recursive session types s_stdout and l_stdout in Figure 8. The protocol defined by those session types requires a client to acquire standard out before being able to print to it and then to release it upon completion. The processes *p* and *v* implement the session types s_stdout and l_stdout, respectively. We have chosen their names in reminiscence of Dijkstra's semaphore operations P and V. The lines of code below demonstrate how a client interacts with atomic standard out for printing, assuming the channel *out* of type s_stdout to be available as a system service:

```
out′ ← acquire out ; out′.enter ;
out′.print ; send out′ "Hello " ; out′.print ; send out′ "shared " ; out′.print ; send out′ "world!" ;
out′.leave ; out ← release out′ ;
```

In session type l_stdout, we take the liberty to use the connective ⊃, a connective introduced in [26, 58] to support value input. The type "string ⊃ l_stdout" describes as session that receives a value of type string and then continues as a session of type l_stdout. Toninho et al. [58] show how to safely integrate a functional layer with a process layer

$$\mathbf{s\_stdout} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{enter} : \mathsf{l\_stdout}\}$$

$$\mathsf{l\_stdout} = \&\{\mathsf{print} : \mathsf{string} \supset \mathsf{l\_stdout},\ \mathsf{leave} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{s\_stdout}\}$$

```
p : {s_stdout}
c ← p =
  c′ ← accept c ;
  case c′ of
  | enter → c′ ← v
```

```
v : {l_stdout}
c′ ← v =
  case c′ of
  | print → x ← recv c′ ;
            print x ;
            c′ ← v
  | leave → c ← detach c′ ;
            c ← p
```

Figure 8: Atomic standard output. The connective $\supset$ denotes value input, an orthogonal concept introduced in [26, 58].

by means of a linear contextual monad. Those results are orthogonal to sharing and generalize to our language. The statement *print* in process v, lastly, abstracts the actual print primitive on a given platform. To prevent races on this primitive, processes p and v are internal, and the only way for users to interact with standard out is via the system service *out*.

4.3 Nondeterminism

Acquire-release introduces *nondeterminism* into our language because it is unknown which client among several clients that acquire a shared process will succeed. We use this property to implement binary nondeterministic choice in our language.

Figure 9 gives the definition of session type coin and its implementing, mutually recursive processes *coin_head* and *coin_tail*. Session type coin indicates which side of the coin is currently facing up. In the implementation each interaction flips the coin to its opposite side.

$$\mathbf{coin} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus \{\mathsf{head} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{coin},\ \mathsf{tail} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathbf{coin}\}$$

```
coin_head : {coin}
c ← coin_head =
  c′ ← accept c ;
  c′.head ;
  c ← detach c′ ;
  c ← coin_tail
```

```
coin_tail : {coin}
c ← coin_tail =
  c′ ← accept c ;
  c′.tail ;
  c ← detach c′ ;
  c ← coin_head
```

Figure 9: Session type coin with implementing processes *coin_head* and *coin_tail*.

Figure 10 shows the process *nd_choice* which nondeterministically sends yes or no and then terminates. Process *nd_choice* achieves nondeterminism by reading a coin that it shares with process *coin_flipper*. Since both processes try to acquire the coin concurrently and the coin switches sides when read, the value read by *nd_choice* depends on the order in which the coin is acquired. For a client of this service, see Figure 11 where it is used to model nondeterminism inherent in the (untyped) asynchronous π-calculus.

5 Semantics

In this section, we complete the discussion of the semantics of $\mathsf{SILL_S}$, by giving the configuration typing rules as well as elaborating on preservation and progress. A complete listing of $\mathsf{SILL_S}$'s abstract syntax, statics, and dynamics as well as proofs of preservation and progress can be found in the appendix. In the last subsection, we sketch an asynchronous dynamics for $\mathsf{SILL_S}$, which relies on a novel transformation derived from logic.

```
nd_choice : {⊕{yes : 1, no : 1}}

d ← nd_choice =
  c ← coin_head ;
  f ← coin_flipper ← c ;
  c′ ← acquire c ;
  case c′ of
  | head → c ← release c′ ; d.yes ; wait f ; close d
  | tail → c ← release c′ ; d.no ; wait f ; close d
```

```
coin_flipper : {1 ← coin}

d ← coin_flipper ← c =
  c′ ← acquire c ;
  case c′ of
  | head → c ← release c′ ; close d
  | tail → c ← release c′ ; close d
```

Figure 10: Binary nondeterministic choice.

5.1 Configuration Typing

At run-time, a $\mathsf{SILL_S}$ program evolves into a number of linear and shared processes as well as placeholders for formerly shared processes that are currently linear. To type the resulting *configuration* Ω, we divide the configuration into a *linear* part Θ and a *shared* part Λ, subject to the following well-formedness conditions:

$$\begin{array}{lll}
\Omega & \triangleq \cdot \mid \Lambda;\, \Theta & (\forall a.\mathsf{proc}(a_{\mathsf{L}},\, \_) \in \Theta \implies \mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda) \\
\Lambda & \triangleq \cdot \mid \mathsf{proc}(a_{\mathsf{S}},\, P_{a_{\mathsf{S}}}),\, \Lambda' \mid \mathsf{unavail}(a_{\mathsf{S}}),\, \Lambda' & (\mathsf{proc}(a_{\mathsf{S}},\, \_),\, \mathsf{unavail}(a_{\mathsf{S}}) \text{ not in } \Lambda') \\
\Theta & \triangleq \cdot \mid \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}),\, \Theta' & (\mathsf{proc}(a_{\mathsf{L}},\, \_) \text{ not in } \Theta')
\end{array}$$

The side conditions make sure that no other process (or placeholder) exists yet in the configuration that provides along the same channel and that for every linear process there exists a placeholder along the shared mode of the channel. The division is justified by the hierarchy between mode S and L, making sure that shared processes cannot depend on linear processes. We use the following typing judgment to type a configuration:

$$\Gamma \vDash_{\Sigma} \Lambda;\, \Theta :: \Gamma;\, \Delta$$

The judgment expresses that the configuration $\Lambda;\, \Theta$ provides the shared channels in Γ and the linear channels in Δ. To permit cyclic dependencies along shared channels, a configuration is type-checked relative to all shared channels, which is the reason why Γ appears to the left of the turnstile. The typing of a configuration is defined by the following rule:

$$\frac{\Gamma \vDash_{\Sigma} \Lambda :: \Gamma \qquad \Gamma \vDash_{\Sigma} \Theta :: \Delta}{\Gamma \vDash_{\Sigma} \Lambda;\, \Theta :: \Gamma;\, \Delta}\ (\text{T-}\Omega)$$

The rule relies on the judgment $\Gamma \vDash_{\Sigma} \Theta :: \Delta$ for typing Θ and the judgment $\Gamma \vDash_{\Sigma} \Lambda :: \Gamma$ for typing Λ. The judgment $\Gamma \vDash_{\Sigma} \Theta :: \Delta$ expresses that the configuration Θ provides the linear channels in Δ, using the shared channels in Γ. The typing of Θ is defined by the following two rules:

$$\frac{}{\Gamma \vDash_{\Sigma} (\cdot) :: (\cdot)}\ (\text{T-}\Theta_1) \qquad \frac{(a_{\mathsf{S}} : \hat{B}) \in \Gamma \qquad \vdash_{\Sigma} (A_{\mathsf{L}}, \hat{B})\ \mathsf{esync} \qquad \Gamma;\, \Delta' \vdash_{\Sigma} P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}}) \qquad \Gamma \vDash_{\Sigma} \Theta : \Delta, \Delta'}{\Gamma \vDash_{\Sigma} \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}),\, \Theta :: (\Delta, a_{\mathsf{L}} : A_{\mathsf{L}})}\ (\text{T-}\Theta_2)$$

Rule (T-Θ_2) is of particular interest as it imposes an order on linear configurations. By requiring that all the linear channels Δ' used by $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}})$ are provided by the remaining configuration Θ, the rule "flattens" the linear process tree such that for any process the providers of the channels used by the process are to the right of the process in the configuration. We maintain this order only for typing purposes, at run-time any permutations of a well-typed configuration are permissible. The rule also enforces that a linear configuration only provides the channels that are not used internally to the configuration. For example, the channels Δ' consumed by $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}})$ are no longer provided as part of the resulting configuration $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}),\, \Theta$. An initial configuration $\Lambda;\, \Theta$ would be typed as $\Gamma \vDash_{\Sigma} \Lambda;\, \Theta :: (\Gamma; c_{\mathsf{L}} : \mathbf{1})$, where the process providing along channel c_{L} is the main program thread and Λ may provide some pre-defined shared system services such a *out* as in Section 4.2. The premises $(a_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\vdash_{\Sigma} (A_{\mathsf{L}}, \hat{B})\ \mathsf{esync}$ of rule (T-Θ_2), constrain the type to which $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}})$ must be released to.

Unlike the typing rules for Θ, the typing rules for Λ do not impose any order on the shared processes. Any attempt would be futile anyway because the reference structure along shared channels may not adhere to any pattern and could, for example, be cyclic. We use the judgment $\Gamma \vDash_\Sigma \Lambda :: \Gamma'$ to type such configurations, expressing that Λ offers the shared channels in Γ', using the shared channels in Γ. The typing rules for Λ are:

$$\frac{}{\Gamma \vDash_\Sigma (\cdot) :: (\cdot)}\ (\text{T-}\Lambda_1) \qquad \frac{\vdash_\Sigma (\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}, \top)\ \mathsf{esync} \qquad \Gamma \vdash_\Sigma P_{a_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})}{\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\, P_{a_{\mathsf{S}}}) :: (a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})}\ (\text{T-}\Lambda_2) \qquad \frac{}{\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}) :: (a_{\mathsf{S}} : \hat{A})}\ (\text{T-}\Lambda_3)$$

$$\frac{\Gamma \vDash_\Sigma \Lambda :: \Gamma' \qquad \Gamma \vDash_\Sigma \Lambda' :: \Gamma''}{\Gamma \vDash_\Sigma \Lambda, \Lambda' :: \Gamma', \Gamma''}\ (\text{T-}\Lambda_4)$$

Rule (T-Λ_4) permits breaking up a configuration Λ into its subparts at any point. Rule (T-Λ_2) carries again an equi-synchronizing invariant as a premise, indicating that the type to which $\mathsf{proc}(a_{\mathsf{S}},\, P_{a_{\mathsf{S}}})$ must be released is not yet significant.

5.2 Preservation and Progress

In this section, we state preservation and progress for $\mathsf{SILL_S}$ and review the key issues that had to be adressed to prove preservation and progress. The proofs of preservation and progress can be found in Section D.

The challenges that arise from extending the linear system discussed in Section 2 with manifest sharing are twofold. For preservation, we need to make sure that clients will encounter shared processes at the type they would like to acquire them. For progress, we need to account for the possibility of deadlock due to cyclic dependencies along shared channels or termination of a process providing a shared service, while ruling out other forms of failure of progress.

To address the first challenge, we have introduced the notion of an equi-synchronizing session type in Section 3.3, which statically imposes the invariant that each shared channel is released to the same session type at which is was acquired (if at all). The preservation proof shows that this invariant is maintained for each channel along any possible transition, as captured in the corresponding premises of rules (T-Θ_2) and (T-Λ_2). Key are the three form of type constraints $\hat{D}$ with $\vdash_\Sigma (A, \hat{D})\ \mathsf{esync}$ where A is the current type of a linear process providing along a_{L}:

1. $\hat{D} = \top$, indicating that there is no constraint on a future $\mathsf{release}$ of a_{L} because a_{L} has never been shared. $\hat{D} = \top$ holds initially, when a linear process is spawned, and continues to hold until the process is released for the first time to become shared. Processes which remain linear throughout their lifetime will never be subject to an equi-synchronization constraint.
2. $\hat{D} = D_{\mathsf{S}}$, indicating that if there is a future release of a_{L} to a shared channel a_{S}, then a_{S} must have type D_{S}. Preservation holds since we have statically checked that $\vdash_\Sigma (A, \hat{D})\ \mathsf{esync}$ and this property is maintained along all continuations of A.
3. $\hat{D} = \bot$, expressing that a_{S} must never be released, which means that any client attempting to acquire a_{S} will be blocked forever. The need for $\bot$ is subtle. Imagine we forward between two linear channels $\mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}$. The forward has to identify not only a_{L} and b_{L}, but also the underlying shared channels a_{S} and b_{S}, because releasing one now amounts to releasing the other:

$$\mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}) \longrightarrow a_{\mathsf{L}} = b_{\mathsf{L}},\, a_{\mathsf{S}} = b_{\mathsf{S}} \qquad (\text{D-I}\text{D}_{\mathsf{L}})$$

   While the types of a_{L} and b_{L} must be at the same A, it is possible that the constraints on the releases of a_{L} and b_{L} are $\vdash (A, D_{\mathsf{S}})\ \mathsf{esync}$ and $\vdash (A, D'_{\mathsf{S}})\ \mathsf{esync}$ for $D_{\mathsf{S}} \neq D'_{\mathsf{S}}$. This can arise because a_{L} and b_{L} may have different histories. Preservation still holds in this case because there can not be a down shift in any continuation of A (shown by coinduction on the definition of esync), so neither a_{L} nor b_{L} could ever be released. Formally, this is conveniently expressed as $\vdash_\Sigma (A, \bot)\ \mathsf{esync}$.

The introduction of $\bot$ requires us to generalize all the typing rules where a process uses a shared channel. For example, we change rule (T-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$L) as follows:

$$\frac{\hat{B} \leq \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}} \qquad \Gamma, x_{\mathsf{S}} : \hat{B};\, \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \hat{B};\, \Delta \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, x_{\mathsf{S}}\, ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\uparrow^{\mathsf{S}}_{\mathsf{L}}\text{L})$$

In contrast to the rule introduced in Section 3.2 the above rule accounts for the possibility of a shared process to be of type $\bot$. In this case, a client can freely choose the type of the process to be acquired because it will never succeed in acquiring that process. As can be seen in Figure 3, the rules (T-$\textsc{Id}_{\mathsf{S}}$), (T-$\textsc{Spawn}_{\mathsf{LL}}$), (T-$\textsc{Spawn}_{\mathsf{LS}}$), (T-$\textsc{Spawn}_{\mathsf{SS}}$), (T-$\exists_{\mathrm{R}}$), and (T-$\Pi_{\mathrm{L}}$) require an analogous treatment.

We can finally state the preservation theorem. It expresses that the types of the providing linear channels are maintained along transitions and that new shared channels may be allocated.

Theorem 1 (Preservation). *If* $\Gamma \vDash_\Sigma \Lambda; \Theta :: \Gamma; \Delta$ *and* $\Lambda; \Theta \longrightarrow \Lambda'; \Theta'$*, then* $\Gamma' \vDash_\Sigma \Lambda'; \Theta' :: \Gamma'; \Delta$*, for some* Λ'*,* Θ'*, and* Γ'*.*

Proof. Preservation is proved by induction on the dynamics, constructing a derivation of a well-typed configuration $\Gamma' \vDash_\Sigma \Lambda''; \Theta'' :: \Gamma'; \Delta$, where Λ'' and Θ'' are permutations of Λ' and Θ', respectively, and using a variety of substitution lemmas and inversion. Note that the linear context Δ remains the same: freshly spawned linear channels have both a provider and client and are therefore not part of the interface. The set of shared channels however can grow. $\square$

Our progress theorem is based on the notion of a *poised* process introduced in [49]. A $\mathsf{proc}(a, P_a)$ is *poised* if it is communicating along its providing channel. The poised forms of processes in $\mathsf{SILL_S}$ are:

Receiving	*Sending*
$\mathsf{proc}(a_{\mathsf{L}},\ y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_y)$	$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b\ ;P)$
	$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}})$
$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P})$	$\mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;P)$
$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;P_{x_{\mathsf{L}}})$	$\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ;P_{x_{\mathsf{S}}})$

A linear configuration Θ is poised if all $\mathsf{proc}(a_{\mathsf{L}}, P_{a_{\mathsf{L}}}) \in \Theta$ are poised and a shared configuration Λ is poised if all $\mathsf{proc}(a_{\mathsf{S}}, P_{a_{\mathsf{S}}}) \in \Lambda$ are poised.

To account for the possibility of deadlock, we introduce the notion of a *blocked* process. We say that a process is *blocked along* a_{S} if it has the form $\mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;Q_{x_{\mathsf{L}}})$. We then state the progress theorem such as to express that being blocked is the *only* way the whole configuration may be stuck [27]. Case (2-c) captures the scenario where a blocked process cannot proceed because the shared channel is unavailable. A successful acquire, on the other hand, is represented as part of case (2-a).

Theorem 2 (Progress). *If* $\Gamma \vDash_\Sigma \Lambda; \Theta :: \Gamma; \Delta$*, then either*

1. $\Lambda \longrightarrow \Lambda'$*, for some* Λ'*, or*
2. Λ *is poised and*
   (a) $\Lambda;\ \Theta \longrightarrow \Lambda';\ \Theta'$*, for some* Λ' *and* Θ'*, or*
   (b) Θ *is poised, or*
   (c) some process in Θ *is blocked along* a_{S} *and* $\mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$*.*

Proof. Progress is proved by induction on the typing of the configurations Λ and Θ. $\square$

At the top level, we have $\Delta = (c_0 : \mathbf{1})$, which means that if Θ is poised then it and all subcomputations must be finished, trying to close c_0. If it cannot transition, then the remaining possibility is that some process in Θ is blocked along a shared channel. A blocked process may wait indefinitely in case of a deadlock, or because the underlying shared process has terminated, or may never be released. Dining philosophers (Figure 7), for instance, is an example leading to a classic deadlock due to a cyclic dependency along the shared forks. Section E.2 gives another example.

5.3 Asynchronous Dynamics

The synchronous operational semantics we have provided for $\mathsf{SILL_S}$ is simple, but not realistic in many applications. Fortunately, we can easily model *asynchronous output* in the existing language in a logically meaningful way. In order to explain this, we reintroduce the general form of cut (spawn) which is not tied to process definitions, and remind the reader of the identity (forward) rule. For simplicity, we restrict the presentation to the linear case; the shown technique directly generalizes to the shared case.

$$\frac{\Delta \vdash P_x :: (x : A) \quad \Delta', x : A \vdash Q_x :: (z : C)}{\Delta, \Delta' \vdash_\Sigma x \leftarrow P_x \; ; \; Q_x :: (z : C)} \; (\text{T-Cut}) \qquad \frac{}{y : A \vdash_\Sigma \mathsf{fwd}\; x\; y :: (x : A)} \; (\text{T-Id})$$

To asynchronously send a channel y along x we spawn a new process which carries the message y, immediately followed by forwarding.

$$\mathsf{send}\; x\; y \; ; \; P \quad \simeq \quad x' \leftarrow (\mathsf{send}\; x\; y \; ; \; \mathsf{fwd}\; x'\; x) \; ; \; [x'/x]P$$

Intuitively, the spawned process ($\mathsf{send}\; x\; y \; ; \; \mathsf{fwd}\; x'\; x$) represents the message y sent along x with fresh continuation channel x' [17]. The continuation channel is necessary so that multiple messages sent along the same channel are guaranteed to arrive in the correct order. It is easy to see that, if the synchronous form on the left is well-typed, then so is the asynchronous form on the right. Logically, we can obtain the proof of the left from the proof of the right by a commuting conversion and reduction of cut with identity.

Operationally, the single synchronous reduction

$$\begin{array}{l} \mathsf{proc}(c,\, \mathsf{send}\; a\; b \; ; \; P),\, \mathsf{proc}(a,\, y \leftarrow \mathsf{recv}\; a \; ; \; Q_y) \\ \quad \longrightarrow \mathsf{proc}(c,\, P),\, \mathsf{proc}(a,\, [b/y]Q_y) \end{array}$$

is now decomposed into several steps, where P can proceed with its continuation before b is received.

$$\begin{array}{ll} \mathsf{proc}(c,\, x' \leftarrow (\mathsf{send}\; a\; b \; ; \; \mathsf{fwd}\; x'\; a) \; ; \; [x'/a]P),\, \mathsf{proc}(a,\, y \leftarrow \mathsf{recv}\; a \; ; \; Q_y) & \\ \quad \longrightarrow \mathsf{proc}(c,\, [a'/a]P),\, \mathsf{proc}(a',\, \mathsf{send}\; a\; b \; ; \; \mathsf{fwd}\; a'\; a),\, \mathsf{proc}(a,\, y \leftarrow \mathsf{recv}\; a \; ; \; Q_y) & (\text{spawn, } a' \text{ fresh}) \\ \quad \longrightarrow \mathsf{proc}(c,\, [a'/a]P),\, \mathsf{proc}(a',\, \mathsf{fwd}\; a'\; a),\, \mathsf{proc}(a,\, [b/y]Q_y) & (\text{receive}) \\ \quad \longrightarrow \mathsf{proc}(c,\, [a'/a]P),\, \mathsf{proc}(a',\, [a'/a][b/y]Q_y) & (\text{forward}) \end{array}$$

Since a' is chosen globally fresh and a is linear, the result is an α-variant of the synchronous outcome. This technique can be applied to all send operations of the semantics. Effectively, this allows a program written in the synchronous style to be executed fully asynchronously.

The caveat is that we would not want to translate acquire in this manner even though the logical semantics dictates it must be a send operation [49]. The reason is that a process would no longer block when trying to acquire a shared channel. Instead it would continue until the corresponding linear channel is actually used to receive a message, which is not the intended meaning. In the implementation (see Section 7) all sends are asynchronous, using a more efficient message buffer instead of explicit continuation channels, except for acquire which blocks until the shared channel becomes available.

Alternatively, we could directly provide an asynchronous semantics for all the operations and use an additional acknowledgment step (a "double shift" [49]) to ensure that acquiring a shared resource is synchronous. For this paper, we have chosen the former route because it simplifies the operational semantics and therefore our theorems: without loss of expressiveness, we do not have to explicitly deal with messages or message queues.

6 Recovering the Computational Power of the Untyped π-calculus

When we view Howard's original isomorphism between typed λ-calculus and intuitionistic natural deduction [32] as a type assignment system for untyped λ-terms, we lose much of the computational power of the untyped λ-calculus. For example, normalization for natural deduction implies termination of computation on well-typed λ-terms, while arbitrary λ-terms may not have a normal form. However, there is a simple way we can embed *all* untyped λ-terms if we add recursive types. In linear instances of the Curry-Howard correspondence, just adding recursion appears insufficient to recover the computational power of the asynchronous π-calculus [62], and so far there has been no logically motivated and fully satisfactory way to do so.[4]

[4]Other recent work in this direction in the setting of classical linear logic by Atkey et al. [2] uses quite different techniques from ours.

In this section, we give an encoding of the asynchronous, untyped π-calculus into $\mathsf{SILL_S}$, demonstrating that shared channels recover the computational power of the untyped π-calculus. The key points to address in the encoding are that *(i)* π-calculus channels may connect arbitrarily many processes, *(ii)* messages sent along a π-calculus channel may arrive in arbitrary order, and *(iii)* π-calculus channels are untyped. Furthermore, since the π-calculus permits deadlock, it is important here that $\mathsf{SILL_S}$ also admits deadlock.

The basic idea of our encoding is to translate π-calculus *processes* to *linear* $\mathsf{SILL_S}$ processes of type $\mathbf{1}$, and π-calculus *channels* to *shared* $\mathsf{SILL_S}$ processes of a universal shared type $\mathcal{U}_\mathsf{S}$. The latter are unordered buffers and obey the following protocol:

$$\begin{aligned}\mathcal{U}_\mathsf{S} = {\uparrow}^\mathsf{S}_\mathsf{L}\, \&\{&\mathsf{ins} : \Pi\mathcal{U}_\mathsf{S}.{\downarrow}^\mathsf{S}_\mathsf{L}\,\mathcal{U}_\mathsf{S},\\ &\mathsf{del} : \oplus\{\mathsf{none} : {\downarrow}^\mathsf{S}_\mathsf{L}\,\mathcal{U}_\mathsf{S},\\ &\qquad\quad \mathsf{some} : \exists\mathcal{U}_\mathsf{S}.{\downarrow}^\mathsf{S}_\mathsf{L}\,\mathcal{U}_\mathsf{S}\}\}\end{aligned}$$

Type $\mathcal{U}_\mathsf{S}$ provides the choice to either send (ins) or receive (del) a channel. In the latter case, it communicates whether the buffer is empty (none) or not empty (some) and delivers a channel in the buffer in the latter case. Figure 11 shows the processes *empty* and elem that implement session type $\mathcal{U}_\mathsf{S}$. To guarantee that the resulting buffer is unordered, process *elem* nondeterministically inserts the received channel at an arbitrary point in the buffer, using *nd_choice* defined in Figure 10. It is also possible and slightly more complicated to postpone the nondeterministic choice to the deletion operation.

```
empty : {U_S}                                 elem : {U_S ← U_S, U_S}

c ← empty =                                   c ← elem ← x, d =
  c′ ← accept c ;                               c′ ← accept c ;
  case c′ of                                    case c′ of
  | ins → x ← recv c′ ;                         | ins → y ← recv c′ ;
          e ← empty ;                                   ndc ← nd_choice ;
          c ← detach c′ ; c ← elem ← x, e               case ndc of
  | del → c′.none ;                                     | yes → e ← elem ← x, d ;
          c ← detach c′ ; c ← empty                             wait ndc ;
                                                                c ← detach c′ ; c ← elem ← y, e
                                                        | no → d′ ← acquire d ;
                                                               d′.ins ; send d′ y ;
                                                               d ← release d′ ;
                                                               wait ndc ;
                                                               c ← detach c′ ; c ← elem ← x, d
                                                | del → c′.some ;
                                                        send c′ x ;
                                                        c ← detach c′ ; fwd c d
```

Figure 11: Processes *empty* and *elem* implement session type $\mathcal{U}_\mathsf{S}$, representing a π-calculus channel. To guarantee that the resulting buffer is unordered, process *elem* nondeterministically inserts the received channel at an arbitrary point in the buffer, using process *nd_choice* defined in Figure 10.

The linear $\mathsf{SILL_S}$ processes representing π-calculus processes now simply amount to "producers" and "consumers" of shared channels of type $\mathcal{U}_\mathsf{S}$. Any number of such processes can communicate along a π-calculus channel by acquiring the shared $\mathsf{SILL_S}$ channel of universal type.

We are now ready to give the encoding of processes. We first review the syntax of the *asynchronous monadic* π-calculus [41, 52], defining the set P^π of π-calculus process terms. We follow the presentation in [3]:

$$P \quad \triangleq \quad \mathsf{0} \;\mid\; \overline{x}\langle y\rangle \;\mid\; x(y).P \;\mid\; \nu x\, P \;\mid\; P_1 \mid P_2 \;\mid\; !P$$

$\mathsf{0}$ denotes an inactive process. $\overline{x}\langle y\rangle$ represents an asynchronous send of y along channel x. $x(y).P$ represents the receiving of a channel along channel x, after which the process continues with executing P with the received channel bound to y in P. The action prefix $x(y)$ acts as a guard, making sure that P can only become active once the input has occurred. $\nu x\, P$ introduces a new channel x that is bound in P. $P_1 \mid P_2$ denotes parallel composition of P_1 and P_2 and $!P$ replication of P.

Our translation shown in Figure 12 yields for each π-calculus process term P^π a corresponding linear process $[\![P^\pi]\!]_a$ in SILL_S, satisfying the typing judgment

$$\Gamma; \cdot \vdash_\Sigma [\![P^\pi]\!]_a :: (a_\mathsf{L} : \mathbf{1})$$

where Γ consists of declarations $x_S : \mathcal{U}_\mathsf{S}$ for every shared channel in the overall process configuration. We use type $\mathbf{1}$ since all communication goes though π-calculus channels, which are mapped to shared channels in Γ. This is also the reason why there are no linear channels in the context. Of course, as shared channels are acquired when send or receive operations are modeled, we communicate with the buffer along a linear channel until it is released again.

$$\begin{array}{lcl}
[\![0]\!]_a & = & \mathsf{close}\ a \\[1ex]
[\![\overline{c}\langle b\rangle]\!]_a & = & p \leftarrow \mathit{snd}\ \mathbf{c}\ ; \\
 & & \mathsf{send}\ p\ \mathbf{b}\ ; \\
 & & \mathsf{wait}\ p\ ; \\
 & & \mathsf{close}\ a \\[1ex]
[\![c(x).P]\!]_a & = & p \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c}\ ; \\
 & & \mathbf{b} \leftarrow \mathsf{recv}\ p\ ; \\
 & & \mathsf{wait}\ p\ ; \\
 & & a \leftarrow [\mathbf{b}/x]\,[\![P]\!]_a \\[1ex]
[\![\nu x\ P]\!]_a & = & \mathbf{e} \leftarrow \mathit{empty}\ ; \\
 & & a \leftarrow [\mathbf{e}/x]\,[\![P]\!]_a \\[1ex]
[\![P_1 \mid P_2]\!]_a & = & b \leftarrow [\![P_1]\!]_b\ ; \\
 & & c \leftarrow [\![P_2]\!]_c\ ; \\
 & & \mathsf{wait}\ b\ ; \\
 & & \mathsf{wait}\ c\ ; \\
 & & \mathsf{close}\ a \\[1ex]
[\![!P]\!]_a & = & \mathit{Rec}^a_{!P} \quad \text{where} \\
\mathit{Rec}^a_{!P} & = & b \leftarrow [\![P]\!]_b\ ; \\
 & & c \leftarrow \mathit{Rec}^c_{!P}\ ; \\
 & & \mathsf{wait}\ b\ ; \\
 & & \mathsf{wait}\ c\ ; \\
 & & \mathsf{close}\ a
\end{array}$$

$$\begin{array}{l}
\mathit{snd} : \{(\Pi\,\mathcal{U}_\mathsf{S}.\mathbf{1}) \leftarrow \mathcal{U}_\mathsf{S}\} \\[1ex]
d \leftarrow \mathit{snd} \leftarrow \mathbf{c} = \\
\quad \mathbf{x} \leftarrow \mathsf{recv}\ d\ ; \\
\quad c' \leftarrow \mathsf{acquire}\ \mathbf{c}\ ; \\
\quad c'.\mathsf{ins}\ ; \\
\quad \mathsf{send}\ c'\ \mathbf{x}\ ; \\
\quad \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\quad \mathsf{close}\ d
\end{array}$$

$$\begin{array}{l}
\mathit{poll\_rcv} : \{(\exists\,\mathcal{U}_\mathsf{S}.\mathbf{1}) \leftarrow \mathcal{U}_\mathsf{S}\} \\[1ex]
d \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c} = \\
\quad c' \leftarrow \mathsf{acquire}\ \mathbf{c}\ ; \\
\quad c'.\mathsf{del}\ ; \\
\quad \mathsf{case}\ c'\ \mathsf{of} \\
\quad \mid \mathsf{none} \rightarrow \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\qquad\qquad\quad d \leftarrow \mathit{poll\_rcv} \leftarrow \mathbf{c} \\
\quad \mid \mathsf{some} \rightarrow \mathbf{x} \leftarrow \mathsf{recv}\ c'\ ; \\
\qquad\qquad\quad \mathbf{c} \leftarrow \mathsf{release}\ c'\ ; \\
\qquad\qquad\quad \mathsf{send}\ d\ \mathbf{x}\ ; \\
\qquad\qquad\quad \mathsf{close}\ d
\end{array}$$

$\mathit{empty} : \{\mathcal{U}_\mathsf{S}\}$ is defined in Figure 11

Figure 12: Translation from untyped asynchronous π-calculus processes to SILL_S and auxiliary processes *snd* and *poll_recv*.

Because of the different semantic basis (asynchronous π-calculus on one hand and multiset rewriting on the other), and the question what precisely is observable about a computation, the precise nature of the correspondence between traces in the source and target is difficult to formulate and prove and left to future work (see Section 9 for further remarks).

7 Implementation

We briefly describe our implementation of manifest sharing in the context of a type-safe C-like imperative language with session types called Concurrent C0 [65], which is an extension of C0 [1, 48] designed for and used in an introductory imperative programming course [50]. Because session-typed programming follows a monadic style, this imperative implementation is semantically adequate for exploration of the expressive power and programming style for manifest sharing. We have transliterated all the examples in this papers into Concurrent C0 and they will be made available with the implementation artifact should this submission be accepted. Besides an occasional illustrative use of imperative language features (e.g., loops in place of recursion, or mutable arrays instead of sequences), the only significant difference is the lack of parametric polymorphism in Concurrent C0. Examples have therefore been modified to use either base types such as `int`, or ad hoc polymorphism in the form of `void*` which engenders tagging of values with their dynamic type to ensure type safety. The implementation uses asynchronous message passing, as described in Section 5.3. Moreover, the downshift modality $\downarrow_{\mathsf{L}}^{\mathsf{S}}$ has no explicit syntax but implicitly precedes every upshift $\uparrow_{\mathsf{L}}^{\mathsf{S}}$. This is adequate since, just as in this paper, the only constructor of shared mode is an upshift so there is no other possible continuation.

The compiler translates C0 source to C. Each logical thread of control is implemented as an operating system thread as provided by the `pthread` library. Message passing is implemented via shared memory. Each channel is therefore a data structure in shared memory that can progress through linear and shared phases. Figure 13 provides a schematic overview of this data structure. While linear, access is shared between a provider and a client. The channel contains a current direction of communication and a message queue implemented as a ring buffer whose size is calculated from the session type. Access to the buffer for send and receive operations is protected by a mutex and associated condition variable. In the shared phase, there will be zero or one provider and an arbitrary number of clients. The channel therefore contains a flag that indicates whether the channel is currently available to be acquired. This flag is turned off when the channel is acquired by one of the clients and remains off until the client has been detached and the provider is ready to accept another client. Access to this flag is protected by a separate mutex and condition variable. The operating system scheduler will then nondeterministically select one of the clients.

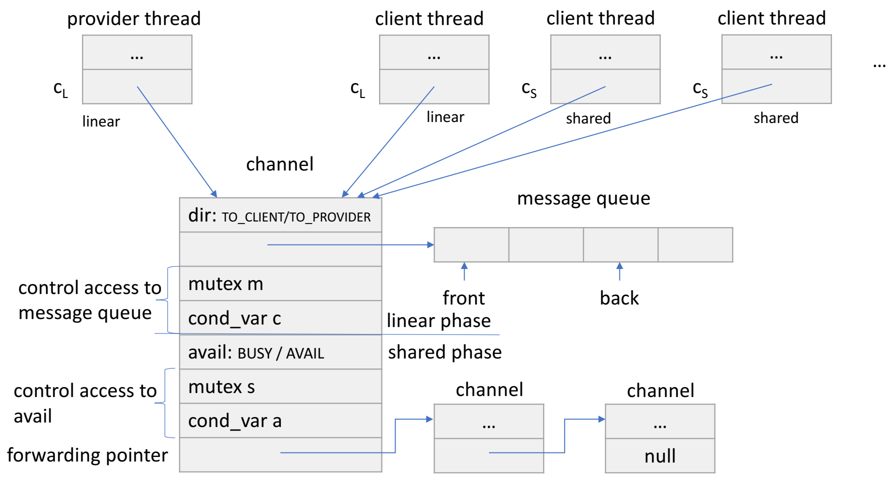

Figure 13: Schematic overview of channel data structure internal to the Concurrent C0 compiler.

As might be expected from the theory, the most difficult aspect of the implementation is forwarding. For forwarding between two linear channels, fwd c d, we send a message `FWD` c along d, or `FWD` d along c, depending on the current direction of communication. Then the thread executing the forward terminates. When the `FWD` e message arrives (where e is either c or d, depending on the direction), the recipient changes its internal reference to the shared channel to e, effectively now continuing communication along e. For more details and some failed alternatives, see [65].

Unfortunately, this strategy fails for forwarding between two shared channels, fwd c d, because there is no effective way to notify all clients of c to now communicate along d via a message. Instead, before terminating, the provider installs a forwarding pointer from c to d and marks the availability of c. Attempts to acquire c will follow the forwarding pointer to d. A potential client may have to follow a whole chain of such forwarding pointers. However, each client has to do so at most once.

Returning to a linear forward: when we execute fwd c d for linear channels c and d that where once shared, the semantics requires that we also forward between the underlying shared channels. For example, if the client replaces references to c by references to d and d is eventually released, then subsequent attempts to acquire c should obtain access to d. In order to account for this scenario, we also install the forwarding pointer from c to d upon a linear forward if the channel has ever been a shared channel with possibly multiple waiting clients.

The current implementation of Concurrent C0 does not deallocate channels that were shared at any point during the program execution. We conjecture that manifest sharing admits an effective reference counting garbage collector by transforming the typing derivation to make implicit applications of weakening and contraction explicit. This is one of the immediately planned items of future work.

8 Related Work

Our work is situated in the family of works on session types [22, 29, 30, 31] among which it extends work based on the Curry-Howard isomorphism between linear logic and session-typed communication [8, 9, 57, 58, 62] with manifest sharing. We have already summarized that work in Section 2 and have pointed out that the shared channels available through the exponential modality in linear logic have a copying semantics and therefore cannot accommodate the examples presented in this paper. Perhaps most closely related is work by Atkey et al. [2] which proceeds by conflating dual pairs of types in classical linear logic, whereas in this paper we maintain the original interpretation of propositions as session types, but provide an alternative operational semantics for a shared layer of channels separated from the linear types by a pair of adjoint modalities.

From the point of view of protocol expression, our work is related to the line of research that uses typestate [56] for protocol checking [5, 15, 20, 40] or program verification [45], in a sequential, object-oriented context. Whereas first approaches [15] support a rather restricted set of aliasing patterns to facilitate modular protocol checking, subsequent approaches lift some of the imposed restrictions, notably by combining aliasing information with typestate [5, 43] or rely-guarantee-based reasoning [40]. Most closely related to our work is Fähndrich's and DeLine's work 2002 on adoption and focus for protocol checking in an object-oriented language. In the resulting language, linear and non-linear objects coexist such that every non-linear object (adoptee) has a linear adopter. Aliases are permitted to adoptees, as long as access goes through the adopter and mutating access happens in a temporary scope, called focus. While an aliased object is in focus, access to the object via another alias is disabled by capability tracking. From this aspect, a focus scope bears ressemblance to a critical section arising between acquire and release points in our system, even though adoption and focus are employed in a purely sequential setting. Whilst capabilities are treated as resources, the underlying type system is not linear, but the required semantics is achieved by threading the capabilities through program execution.

From the point of view of allowing controlled aliasing in a concurrent setting, our work is related to permission-based logics [6, 28, 39, 54] and concurrent separation logic [7, 36, 46, 59, 60]. Permission-based logics maintain a distinction between read and write access to a shared memory location, allowing read access even if only a fractional permission [6] is held, whereas write access requires the entire permission. From a session type perspective, this distinction is less relevant because any communication, input (write) and output (read) alike, amounts to a change in protocol state and thus must be protected sufficiently. Separation logic shares with linear logic the separating conjunction to reason about resource consumption, but uses a Hoare-style reasoning approach that is extrinsic to the type system, whereas resource-awareness is intrinsic to our type system via the Curry-Howard correspondence. Moroever, both permission-based logics and concurrent separation logic target shared-memory concurrency, whereas our work is situated in the realm of message-passing concurrency, offering a different level of abstraction.

Linear types have also found various applications in systems programming. Walker and Watkins 2001, for example, combine linear types with regions [23], and Smith et al. relax the operational "use-once" semantics of linear types [61] to exploit pointer aliasing for destructive operations. Similar observations have been made by Castegren and Wrigstad 2016 in the context of implementing lock-free algorithms. Our work differs from these approaches in that it is based on a richer semantics of linearity derived from the Curry-Howard isomorphism between linear logic and session-typed communication. Moreover, our work employs a message-passing approach to concurrency rather than a shared-memory-based approach. From this perspective, our work has closer ties with the Rust systems programming language [42], which supports message-passing concurrency in an affine setting. Shared data in Rust is normally immutable, but Rust also supports various abstractions (e.g., mutexes) that support the safe mutation of shared data. We have found that the programming patterns arising in $\mathsf{SILL_S}$ readily translate into Rust code with mutexes. Rust mutexes, however, are dynamic notions only, and Rust does furthermore not support protocol expression.

9 Discussion and Future Work

We have presented an extension of logic-based session-typed message-passing concurrency by permitting shared resources encapsulated in processes. This allows elegant expression of examples such as queues with multiple producers and multiple consumers, dining philosophers, shared databases, shared input and output devices, or nondeterministic choice. In fact, all of the asynchronous π-calculus can now be embedded in a statically typed framework satisfying session fidelity by modeling π-calculus channels as shared processes maintaining a nondeterministic message buffer. We were able to maintain the view of linear propositions as session types, sequent proofs as processes, and linear proof reduction as communication. To accomodate shared processes, we had to generalize the usual Curry-Howard correspondence and allow interleaved proof construction (acquire), proof reduction (communication), and proof deconstruction (release). Proof construction may fail, which manifests operationally as deadlock. Key insights are the decomposition of the exponential modality $!A$ into $\downarrow_{\mathsf{L}}^{\mathsf{S}}\uparrow_{\mathsf{L}}^{\mathsf{S}}A_{\mathsf{L}}$ inspired by adjoint logic and the insistence on equi-synchronizing types which guarantees that a shared process is always released at the same type it was acquired. The former makes sharing manifest in the type; the latter guarantees session fidelity without runtime checking of types.

On the theory side, we plan to consider how to overlay a likely very different type system or static analysis in order to recover absence of deadlocks. Some recent promising work in this direction [37, 38] in a different context may be adaptable to our situation. We are also interested in relaxing the restriction on equi-synchronization. A first avenue to pursue is to extend our definitions to support subtyping, along the lines of Gay and Hole [22]. Another possibility is to complement the static approach with run-time type-checking to maintain session fidelity [35], particularly in a distributed setting. On the implementation side, we would like to develop the proof-theoretic foundation of a reference counting implementation so that resources associated with shared processes that are no longer accessible can be released.

Finally, the embedding of the asynchronous π-calculus into $\mathsf{SILL}_{\mathsf{S}}$ raises the interesting question of how precise the modeling is. While we can easily relate computation traces, other traditional notions of concurrency theory such as bisimulation do not immediately apply since our semantics is given as a multiset rewriting system. We conjecture that a slightly modified interpretation with late application of nondeterministic choice describes a bisimulation, according to the definitions mapped out by Deng et al. 2016.

A Abstract Syntax

Table 3 defines the abstract syntax of $\mathsf{SILL_S}$. As is usual, we use the overline-notation to denote a sequence. For example, $\oplus\{\overline{l : A_\mathsf{L}}\}$ stands for $\oplus\{l_1 : A_{\mathsf{L}\,1}, \ldots, l_n : A_{\mathsf{L}\,n}\}$. By convention, lines without a left-hand side are separated by $|$ from their preceding line.

Sort		Abstract Form	Remarks
Metavariables	$\triangleq$	x_m, y_m, z_m	variable (or channel)
		a_m, b_m, c_m, d_m	channel
		l	label
Modes m	$\triangleq$	$\mathsf{S} \mid \mathsf{L}$	with $\mathsf{S} > \mathsf{L}$
Constraint $\hat{A}, \hat{B}, \hat{C}, \hat{D}$	$\triangleq$	$\top$	no yet detached
		$\bot$	can no longer be acquired
		$\uparrow^\mathsf{S}_\mathsf{L} A_\mathsf{L}$	constraint on which type must be released to
Session types A_m, B_m, C_m, D_m	$\triangleq$	$\oplus\{\overline{l : A_\mathsf{L}}\}$	internal choice, at least one label
		$\&\{\overline{l : A_\mathsf{L}}\}$	external choice, at least one label
		$A_\mathsf{L} \otimes B_\mathsf{L}$	linear channel output
		$A_\mathsf{L} \multimap B_\mathsf{L}$	linear channel input
		$\exists A_\mathsf{S}.B_\mathsf{L}$	shared channel output
		$\Pi A_\mathsf{S}.B_\mathsf{L}$	shared channel input
		$\mathbf{1}_\mathsf{L}$	termination
		$\downarrow^\mathsf{S}_\mathsf{L} A_\mathsf{S}$	downshift
		$\uparrow^\mathsf{S}_\mathsf{L} A_\mathsf{L}$	upshift
		Y_m	type variable
Definition X_m, Y_m	$\triangleq$	$x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m,\overline{y}}$	process definition
		$Y_m = A_m$	recursive session type definition
Process P, Q	$\triangleq$	$x_\mathsf{L}.l_h \,; P$	label output
		$\mathsf{case}\; x_\mathsf{L}\; \mathsf{of}\; \overline{l \Rightarrow P}$	label input
		$\mathsf{send}\; x_\mathsf{L}\; y_m \,; P$	channel output
		$y_m \leftarrow \mathsf{recv}\; x_\mathsf{L} \,; P_{y_m}$	channel input
		$\mathsf{close}\; x_\mathsf{L}$	terminate process
		$\mathsf{wait}\; x_\mathsf{L} \,; Q$	wait for process to terminate
		$\mathsf{fwd}\; x_m\; y_m$	forward x to y
		$x_m \leftarrow X_m \leftarrow \overline{y} \,; Q_{x_m}$	spawn
		$x_\mathsf{S} \leftarrow \mathsf{detach}\; x_\mathsf{L} \,; P_{x_\mathsf{S}}$	detach (i.e., shift for $\downarrow^\mathsf{S}_\mathsf{L}\mathsf{R}$)
		$x_\mathsf{S} \leftarrow \mathsf{release}\; x_\mathsf{L} \,; Q_{x_\mathsf{S}}$	release (i.e., shift for $\downarrow^\mathsf{S}_\mathsf{L}\mathsf{L}$)
		$x_\mathsf{L} \leftarrow \mathsf{acquire}\; x_\mathsf{S} \,; Q_{x_\mathsf{L}}$	acquire (i.e., shift for $\uparrow^\mathsf{S}_\mathsf{L}\mathsf{L}$)
		$x_\mathsf{L} \leftarrow \mathsf{accept}\; x_\mathsf{S} \,; P_{x_\mathsf{L}}$	accept (i.e., shift for $\uparrow^\mathsf{S}_\mathsf{L}\mathsf{R}$)

Table 3: Abstract syntax.

B Statics

B.1 Signature Checking

A well-formed *signature* Σ consists of a finite set of process definitions and recursive session type definitions. A process definition $x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m,\overline{y}}$ associates a name X with a process term P and indicates the name and type of the process' providing channel and the names and types of its argument channels. A recursive session type definition $Y_m = A_m$ associates a name Y with a type term A:

$$\begin{array}{rcll} \Sigma & \triangleq & \cdot & \\ & | & x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m,\overline{y}},\ \Sigma' & (X_m \text{ not in } \Sigma') \\ & | & Y_m = A_m,\ \Sigma' & (Y_m \text{ not in } \Sigma') \end{array}$$

We assume that the signature Σ is populated prior to type-checking. Each process definition $x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m,\overline{y}}$ gives rise to a corresponding persistent predicate $!\mathsf{def}(x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m,\overline{y}})$ with variables x_m and $\overline{y}$ bound by P that are subject to α-variance to guarantee freshness. We use the subscript notation to separate the linear $\overline{y_{\mathsf{L}}}$ from the shared $\overline{y_{\mathsf{S}}}$ part of the arguments $\overline{y}$.

The rules for type-checking the signature are given in Figure 14, Figure 15, and Figure 16. For recursive session types, we adopt the *equi-recursive* [14] interpretation developed by Gay and Hole 2005, requiring recursive session types to be *contractive*. This requirement is stipulated in rule (T-Σ_3) and defined by the rules shown in Figure 15. Moreover, we require session types to be *equi-synchronizing*, a property we introduce in this paper (see Section 3.3) to guarantee that a process is *released* to the *same* type at which it was *acquired*. The property of equi-synchronizing establishes session fidelity without the need for run-time checks at acquisition points.

$$\frac{}{\vdash_\Sigma (\cdot)\ \ \mathsf{sig}}\ (\text{T-}\Sigma_1)$$

$$\frac{\overline{y_{\mathsf{S}} : B_{\mathsf{S}}};\ \overline{y_{\mathsf{L}} : B_{\mathsf{L}}} \vdash_\Sigma P_{x_m,\overline{y}} :: (x_m : A_m) \qquad \vdash_\Sigma (A_m, \top)\ \mathsf{esync} \qquad \vdash_\Sigma \Sigma'\ \mathsf{sig}}{\vdash_\Sigma (x_m : A_m \leftarrow X_m \leftarrow \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}, \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x_m,\overline{y}}),\ \Sigma'\ \ \mathsf{sig}}\ (\text{T-}\Sigma_2)$$

$$\frac{\vdash_\Sigma A_m\ \ \mathsf{contr} \qquad \vdash_\Sigma (A_m, \top)\ \mathsf{esync} \qquad \vdash_\Sigma \Sigma'\ \mathsf{sig}}{\vdash_\Sigma Y_m = A_m,\ \Sigma'\ \ \mathsf{sig}}\ (\text{T-}\Sigma_3)$$

Figure 14: Signature checking.

$$\frac{}{\vdash_\Sigma \oplus\{\overline{l : A_{\mathsf{L}}}\}\ \ \mathsf{contr}}\ (\text{T-Contr}_\oplus) \qquad \frac{}{\vdash_\Sigma \&\{\overline{l : A_{\mathsf{L}}}\}\ \ \mathsf{contr}}\ (\text{T-Contr}_\&) \qquad \frac{}{\vdash_\Sigma A_{\mathsf{L}} \otimes B_{\mathsf{L}}\ \ \mathsf{contr}}\ (\text{T-Contr}_\otimes)$$

$$\frac{}{\vdash_\Sigma A_{\mathsf{L}} \multimap B_{\mathsf{L}}\ \ \mathsf{contr}}\ (\text{T-Contr}_\multimap) \qquad \frac{}{\vdash_\Sigma \exists A_{\mathsf{S}}.B_{\mathsf{L}}\ \ \mathsf{contr}}\ (\text{T-Contr}_\exists) \qquad \frac{}{\vdash_\Sigma \Pi A_{\mathsf{S}}.B_{\mathsf{L}}\ \ \mathsf{contr}}\ (\text{T-Contr}_\Pi)$$

$$\frac{}{\vdash_\Sigma \mathbf{1}\ \ \mathsf{contr}}\ (\text{T-Contr}_\mathbf{1}) \qquad \frac{}{\vdash_\Sigma \downarrow^r_m A_r\ \ \mathsf{contr}}\ (\text{T-Contr}_{\downarrow^r_m}) \qquad \frac{}{\vdash_\Sigma \uparrow^r_m A_m\ \ \mathsf{contr}}\ (\text{T-Contr}_{\uparrow^r_m})$$

Figure 15: Contractive recursive session type.

B.2 Process Typing

To type *process terms*, we use the judgments:

$$\Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$$

$$\Gamma;\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$$

The judgments rely on the signature Σ and on the contexts Γ and Δ to type shared and linear channels, respectively. The judgment indicates that the process P provides a service of session type A_m along channel x_m, given the typing of services provided by processes along the channels in Δ (and Γ). Since channels are substituted for variables at run-time, we allow the metavariables x_m, y_m, z_m to stand for both variables and channels.

The context Γ is a *structural* context that consists of a finite set of assumptions of the form $x_{\mathsf{S}\,i} : A_{\mathsf{S}\,i}$, associating with each shared variable or channel a session type. A *well-formed* structural context is defined as follows:

$$\Gamma \triangleq \cdot \mid \Gamma',\ x_{\mathsf{S}} : \hat{A} \qquad (x_{\mathsf{S}} \text{ not in } \Gamma')$$

$$\frac{(\forall i)\ \vdash_\Sigma (A_{\mathsf{L}_i}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (\oplus\{\overline{l : A_\mathsf{L}}\}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\oplus) \qquad \frac{(\forall i)\ \vdash_\Sigma (A_{\mathsf{L}_i}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (\&\{\overline{l : A_\mathsf{L}}\}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\&)$$

$$\frac{\vdash_\Sigma (B_\mathsf{L}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (A_\mathsf{L} \otimes B_\mathsf{L}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\otimes) \qquad \frac{\vdash_\Sigma (B_\mathsf{L}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (A_\mathsf{L} \multimap B_\mathsf{L}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\multimap)$$

$$\frac{\vdash_\Sigma (B_\mathsf{L}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (\exists A_\mathsf{S}.B_\mathsf{L}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\exists) \qquad \frac{\vdash_\Sigma (B_\mathsf{L}, \hat{D})\ \mathsf{esync}}{\vdash_\Sigma (\Pi A_\mathsf{S}.B_\mathsf{L}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\Pi)$$

$$\frac{}{\vdash_\Sigma (\mathbf{1}, \hat{D})\ \mathsf{esync}}\ (\text{T-Esync}_\mathbf{1})$$

$$\frac{\vdash_\Sigma (A_\mathsf{L}, \uparrow^\mathsf{S}_\mathsf{L} A_\mathsf{L})\ \mathsf{esync}}{\vdash_\Sigma (\uparrow^\mathsf{S}_\mathsf{L} A_\mathsf{L}, \top)\ \mathsf{esync}}\ (\text{T-Esync}_{\uparrow^\mathsf{S}_\mathsf{L}}) \qquad \frac{\vdash_\Sigma (D_\mathsf{S}, \top)\ \mathsf{esync}}{\vdash_\Sigma (\downarrow^\mathsf{S}_\mathsf{L} D_\mathsf{S}, D_\mathsf{S})\ \mathsf{esync}}\ (\text{T-Esync}_{\downarrow^\mathsf{S}_\mathsf{L}}\text{-1}) \qquad \frac{\vdash_\Sigma (D_\mathsf{S}, \top)\ \mathsf{esync}}{\vdash_\Sigma (\downarrow^\mathsf{S}_\mathsf{L} D_\mathsf{S}, \top)\ \mathsf{esync}}\ (\text{T-Esync}_{\downarrow^\mathsf{S}_\mathsf{L}}\text{-2})$$

Figure 16: Equi-synchronizing session type, coinductively defined.

The side-condition makes sure that the variable or channel name is unique within the context. When concatenating a context Γ with a context Γ', variables or channels $x_\mathsf{S} : \hat{A}$ that are shared between Γ and Γ' are contracted, i.e., reduced to one occurrence.

The context Δ, on the other hand, is a *linear* context that consists of a finite set of assumptions of the form $x_{\mathsf{L}\,i} : A_{\mathsf{L}\,i}$, associating with each linear variable or channel a session type. A *well-formed* linear context is defined as follows:

$$\Delta \triangleq \cdot \mid \Delta',\ x_\mathsf{L} : A_\mathsf{L} \qquad (x_\mathsf{L} \text{ not in } \Delta')$$

Again, the side-condition makes sure that the variable or channel name is unique within the context. We define the projections $\mathsf{dom}(\Gamma)$ and $\mathsf{dom}(\Delta)$ to project onto the variable or channel names in Γ and Δ, respectively, yielding the empty set in case the context is empty.

The well-formedness of a process typing judgment imposes slightly different invariants on variables and channels. Whereas variables are always unique by α-conversion, uniqueness holds only for linear channels, but not shared channels. For a linear process typing judgment to be well-formed, the following condition must hold:

$$\Gamma; \Delta \vdash_\Sigma P :: (x_\mathsf{L} : A_\mathsf{L}) \qquad (x_\mathsf{L} \text{ not in } \Delta)$$

The same condition holds only for shared variables in shared process typing judgments:

$$\Gamma \vdash_\Sigma P :: (x_\mathsf{S} : A_\mathsf{S}) \qquad (x_\mathsf{S} \text{ not in } \Gamma, \text{ if } x_\mathsf{S} \text{ is a variable})$$

Since the same shared channel can be substituted for different shared variables, giving rise to the possibility of cycles among shared channels (see Section E.2), the offering channel of a shared process can already occur in its shared context. We point out that no ambiguity arises in that case between the provision and use of a shared channel because $\mathsf{SILL_S}$ introduces separate constructs for each direction. For example, a_S in $x_\mathsf{L} \leftarrow \mathsf{accept}\, a_\mathsf{S}\,; P_{x_\mathsf{L}}$ denotes the offering channel whereas it denotes a used channel in $x_\mathsf{L} \leftarrow \mathsf{acquire}\, a_\mathsf{S}\,; Q_{x_\mathsf{L}}$.

The typing rules for process terms are given in Figure 17 and Figure 18. Due to the difference in invariants shared variables and channels have to obey, a context of the form $\Gamma, x_\mathsf{S} : \hat{A}$ conveys the information that x_S is fresh, if x is a variable.

B.3 Configuration Typing

Figure 19 gives the typing rules for a $\mathsf{SILL_S}$ *configuration* Ω. A configuration Ω is divided into a a *linear* part Θ and a *shared* part Λ, subject to the following well-formedness conditions:

$$\frac{}{\Gamma;\, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} \mathsf{fwd}\; x_{\mathsf{L}}\, y_{\mathsf{L}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}\;(\text{T-Id}_{\mathsf{L}}) \qquad \frac{\hat{A} \leq A_{\mathsf{S}}}{\Gamma, y_{\mathsf{S}} : \hat{A} \vdash_{\Sigma} \mathsf{fwd}\; x_{\mathsf{S}}\, y_{\mathsf{S}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}\;(\text{T-Id}_{\mathsf{S}})$$

$$\frac{\begin{array}{c}\Gamma = \overline{w_{\mathsf{S}} : \hat{B}} \qquad \overline{\hat{B}} \leq \overline{B_{\mathsf{S}}} \qquad \Delta = \overline{y_{\mathsf{L}} : B_{\mathsf{L}}} \\ (x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}' : B_{\mathsf{L}}}, \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}'}, \overline{w_{\mathsf{S}}'}}) \in \Sigma \qquad \Gamma, \Gamma';\, \Delta', x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})\end{array}}{\Gamma, \Gamma';\, \Delta, \Delta' \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}\,;\; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\;(\text{T-Spawn}_{\mathsf{LL}})$$

$$\frac{\Gamma = \overline{y_{\mathsf{S}} : \hat{B}} \qquad \overline{\hat{B}} \leq \overline{B} \qquad (x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}' : B} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}'}}) \in \Sigma \qquad \Gamma, \Gamma', x_{\mathsf{S}} : A_{\mathsf{S}};\, \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, \Gamma';\, \Delta \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}}\,;\; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\;(\text{T-Spawn}_{\mathsf{LS}})$$

$$\frac{\Gamma = \overline{y_{\mathsf{S}} : \hat{B}} \qquad \overline{\hat{B}} \leq \overline{B} \qquad (x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}' : B} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}'}}) \in \Sigma \qquad \Gamma, \Gamma', x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{S}} : C_{\mathsf{S}})}{\Gamma, \Gamma' \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}}\,;\; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{S}} : C_{\mathsf{S}})}\;(\text{T-Spawn}_{\mathsf{SS}})$$

Figure 17: Process typing for identity and spawn (cut).

$$\begin{array}{rcll}
\Omega & \triangleq & \cdot \mid \Lambda;\, \Theta & (\forall a.\mathsf{proc}(a_{\mathsf{L}},\, \_) \in \Theta \implies \mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda) \\
\Lambda & \triangleq & \cdot \mid \mathsf{proc}(a_{\mathsf{S}},\, P_{a_{\mathsf{S}}}),\, \Lambda' \mid \mathsf{unavail}(a_{\mathsf{S}}),\, \Lambda' & (\mathsf{proc}(a_{\mathsf{S}},\, \_),\, \mathsf{unavail}(a_{\mathsf{S}}) \text{ not in } \Lambda') \\
\Theta & \triangleq & \cdot \mid \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}),\, \Theta' & (\mathsf{proc}(a_{\mathsf{L}},\, \_) \text{ not in } \Theta')
\end{array}$$

We provide further explanations on the configuration typing in Section 5.1, but point out that Rule (T-Ω) permits cyclic dependencies along shared channels by type-checking a configuration relative to all shared channels. Moreover, the rules adhere to the convention to use the metavariables x, y, and z to stand for variables and the metavariables a, b, c, and d to stand for channels. Channels are run-time artifacts, and thus it is known at this stage whether a metavariable denotes a variable or a channel.

C Operational Semantics

Figure 20 gives the operational semantics of $\mathsf{SILL_S}$ using *multiset rewriting rules* [11]. The semantics is *synchronous*, Section 5.3 sketches how to transform this synchronous semantics into an asynchronous one using cut and identity.

A multiset rewriting rule is of the form

$$S_1,\, \ldots,\, S_n \longrightarrow T_1,\, \ldots,\, T_m$$

indicating that the state consisting of $S_1,\, \ldots,\, S_n$ transitions to the one consisting of $T_1,\, \ldots,\, T_m$. The $'\longrightarrow'$ denotes the state transition, substates are separated by $','$. The rule only mentions the part of the state that it rewrites.

We use the following predicates to denote states:

- $\mathsf{proc}(a_m,\, P_m)$, for a process providing along channel a_m and executing the process term P_m;
- $\mathsf{unavail}(a_{\mathsf{S}})$, as a placeholder for a shared process providing along channel a_{S} that is currently not available;
- $!\mathsf{def}(x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m, \overline{y}})$, for each $(x_m : A_m \leftarrow X_m \leftarrow \overline{y : B} = P_{x_m, \overline{y}}) \in \Sigma$.

Even though multiset rewriting rules are unordered, we write the rules such that a providing process appears to the right of its client, for ease of reading.

We use the side-condition (b *fresh*) in

$$S \longrightarrow T \qquad (b \textit{ fresh})$$

to allocate a globally *fresh* channel. The freshly allocated channel may occur in T but not yet in S.

$$\dfrac{\hat{B} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \qquad \Gamma, x_{\mathsf{S}} : \hat{B};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \hat{B};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, x_{\mathsf{S}} \,; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\text{L})
\qquad
\dfrac{\Gamma;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\, x_{\mathsf{S}} \,; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\text{T-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\text{R})$$

$$\dfrac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\, x_{\mathsf{L}} \,; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\text{L})
\qquad
\dfrac{\Gamma \vdash_{\Sigma} P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{detach}\, x_{\mathsf{L}} \,; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})}\ (\text{T-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\text{R})$$

$$\dfrac{\Gamma;\ \Delta \vdash_{\Sigma} Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \mathbf{1} \vdash_{\Sigma} \mathsf{wait}\, x_{\mathsf{L}} \,; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\mathbf{1}_{\text{L}})
\qquad
\dfrac{}{\Gamma;\ \cdot \vdash_{\Sigma} \mathsf{close}\, x_{\mathsf{L}} :: (x_{\mathsf{L}} : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\dfrac{\Gamma;\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{y_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}} \vdash_{\Sigma} y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}} \,; Q_{y_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\otimes_{\text{L}})
\qquad
\dfrac{\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{L}} \,; P :: (x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})}\ (\text{T-}\otimes_{\text{R}})$$

$$\dfrac{\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_{\Sigma} Q_{y_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_{\Sigma} y_{\mathsf{S}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}} \,; Q_{y_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\exists_{\text{L}})
\qquad
\dfrac{\hat{A} \leq A_{\mathsf{S}} \qquad \Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta \vdash_{\Sigma} \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{S}} \,; P :: (x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}))}\ (\text{T-}\exists_{\text{R}})$$

$$\dfrac{\Gamma;\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_{\Sigma} Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{L}} \,; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\multimap_{\text{L}})
\qquad
\dfrac{\Gamma;\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta \vdash_{\Sigma} y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}} \,; P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}})}\ (\text{T-}\multimap_{\text{R}})$$

$$\dfrac{\hat{A} \leq A_{\mathsf{S}} \qquad \Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta, x_{\mathsf{L}} : B_{\mathsf{L}} \vdash_{\Sigma} Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, y_{\mathsf{S}} : \hat{A};\ \Delta, x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_{\Sigma} \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{S}} \,; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\Pi_{\text{L}})
\qquad
\dfrac{\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})}{\Gamma;\ \Delta \vdash_{\Sigma} y_{\mathsf{S}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}} \,; P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}))}\ (\text{T-}\Pi_{\text{R}})$$

$$\dfrac{(\forall i)\ \ \Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}_i} \vdash_{\Sigma} Q_i :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\} \vdash_{\Sigma} \mathsf{case}\, x_{\mathsf{L}}\, \mathsf{of}\, \overline{l \Rightarrow Q} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\oplus_{\text{L}})
\qquad
\dfrac{\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}\,h})}{\Gamma;\ \Delta \vdash_{\Sigma} x_{\mathsf{L}}.l_h \,; P :: (x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})}\ (\text{T-}\oplus_{\text{R}})$$

$$\dfrac{\Gamma;\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}\,h} \vdash_{\Sigma} Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\} \vdash_{\Sigma} x_{\mathsf{L}}.l_h \,; Q :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\&_{\text{L}})
\qquad
\dfrac{(\forall i)\ \ \Gamma;\ \Delta \vdash_{\Sigma} P_i :: (x_{\mathsf{L}} : A_{\mathsf{L}_i})}{\Gamma;\ \Delta \vdash_{\Sigma} \mathsf{case}\, x_{\mathsf{L}}\, \mathsf{of}\, \overline{l \Rightarrow P} :: (x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})}\ (\text{T-}\&_{\text{R}})$$

Figure 18: Process typing for shifts and propositional rules.

Lastly, we use the equation $a = b$ in

$$S \longrightarrow T,\ a = b$$

for the global substitution of b for a in the entire post-configuration. The channels a and b may occur in S.

$$\frac{}{\Gamma \vDash_\Sigma (\cdot) :: (\cdot)}\ (\text{T-}\Theta_1)$$

$$\frac{(a_{\mathsf{S}} : \hat{B}) \in \Gamma \qquad \vdash_\Sigma (A_{\mathsf{L}}, \hat{B})\ \mathsf{esync} \qquad \Gamma;\ \Delta' \vdash_\Sigma P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}}) \qquad \Gamma \vDash_\Sigma \Theta : \Delta, \Delta'}{\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta :: (\Delta, a_{\mathsf{L}} : A_{\mathsf{L}})}\ (\text{T-}\Theta_2)$$

$$\frac{}{\Gamma \vDash_\Sigma (\cdot) :: (\cdot)}\ (\text{T-}\Lambda_1)$$

$$\frac{\vdash_\Sigma (\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}, \top)\ \mathsf{esync} \qquad \Gamma \vdash_\Sigma P_{a_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}{\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ P_{a_{\mathsf{S}}}) :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\text{T-}\Lambda_2)$$

$$\frac{}{\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}) :: (a_{\mathsf{S}} : \hat{A})}\ (\text{T-}\Lambda_3)$$

$$\frac{\Gamma \vDash_\Sigma \Lambda :: \Gamma' \qquad \Gamma \vDash_\Sigma \Lambda' :: \Gamma''}{\Gamma \vDash_\Sigma \Lambda, \Lambda' :: \Gamma', \Gamma''}\ (\text{T-}\Lambda_4)$$

$$\frac{\Gamma \vDash_\Sigma \Lambda :: \Gamma \qquad \Gamma \vDash_\Sigma \Theta :: \Delta}{\Gamma \vDash_\Sigma \Lambda;\ \Theta :: \Gamma;\ \Delta}\ (\text{T-}\Omega)$$

Figure 19: Configuration typing.

$$\begin{array}{l}\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}})\\ \longrightarrow\ a_{\mathsf{L}} = b_{\mathsf{L}},\ a_{\mathsf{S}} = b_{\mathsf{S}}\end{array} \qquad (\text{D-I}\textsc{d}_{\mathsf{L}})$$

$$\begin{array}{l}\mathsf{proc}(a_{\mathsf{S}},\ \mathsf{fwd}\ a_{\mathsf{S}}\ b_{\mathsf{S}})\\ \longrightarrow\ \mathsf{unavail}(a_{\mathsf{S}}),\ a_{\mathsf{S}} = b_{\mathsf{S}}\end{array} \qquad (\text{D-I}\textsc{d}_{\mathsf{S}})$$

$$\begin{array}{l}\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{c_{\mathsf{L}}}, \overline{c_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{L}}}),\ !\mathsf{def}(x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}, \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}}, \overline{y_{\mathsf{S}}}})\\ \longrightarrow\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}]Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}}, \overline{y_{\mathsf{S}}}}),\ \mathsf{unavail}(b_{\mathsf{S}}) \quad (b\ \textit{fresh})\end{array} \qquad (\text{D-S}\textsc{pawn}_{\mathsf{LL}})$$

$$\begin{array}{l}\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{S}}}),\ !\mathsf{def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}})\\ \longrightarrow\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}) \quad (b\ \textit{fresh})\end{array} \qquad (\text{D-S}\textsc{pawn}_{\mathsf{LS}})$$

$$\begin{array}{l}\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\ ;\ Q_{x_{\mathsf{S}}}),\ !\mathsf{def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}})\\ \longrightarrow\ \mathsf{proc}(a_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}) \quad (b\ \textit{fresh})\end{array} \qquad (\text{D-S}\textsc{pawn}_{\mathsf{SS}})$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{release}\ a_{\mathsf{L}}\ ;\ Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ;\ P_{x_{\mathsf{S}}}),\ \mathsf{unavail}(a_{\mathsf{S}})\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\ Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{S}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\ P_{x_{\mathsf{S}}})\end{array} \qquad (\text{D-}{\downarrow}^{\mathsf{S}}_{\mathsf{L}} - \mathsf{release/detach})$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}})\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\ Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\ P_{x_{\mathsf{L}}}),\ \mathsf{unavail}(a_{\mathsf{S}})\end{array} \qquad (\text{D-}{\uparrow}^{\mathsf{S}}_{\mathsf{L}} - \mathsf{acquire/accept})$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}})\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ Q)\end{array} \qquad (\text{D-}\mathbf{1})$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_y),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b\ ;\ P)\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ [b/y]\ Q_y),\ \mathsf{proc}(a_{\mathsf{L}},\ P)\end{array} \qquad (\text{D-}\otimes/\exists)$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ P_y)\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b/y]P_y)\end{array} \qquad (\text{D-}\multimap/\Pi)$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;\ P)\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P)\end{array} \qquad (\text{D-}\oplus)$$

$$\begin{array}{l}\mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P})\\ \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h)\end{array} \qquad (\text{D-}\&)$$

Figure 20: Multiset rewriting system defining synchronous operational dynamics.

D Preservation and Progress

D.1 Definitions, Lemmas, and Corollaries

D.1.1 Definitions

Definition 1 (Configuration Substitution). *Substitution in configurations is defined as follows:*

$$\begin{aligned}
[b_{\mathsf{L}}/a_{\mathsf{L}}]\,(\cdot) &\triangleq (\cdot)\\
[b_{\mathsf{L}}/a_{\mathsf{L}}]\,(\mathsf{proc}(c_{\mathsf{L}},\ Q),\Theta) &\triangleq \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/a_{\mathsf{L}}]\,Q),[b_{\mathsf{L}}/a_{\mathsf{L}}]\,\Theta \qquad \textit{(where } a_{\mathsf{L}} \neq c_{\mathsf{L}}\textit{)}\\
\\
[b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\cdot) &\triangleq (\cdot)\\
[b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\mathsf{proc}(c_{\mathsf{L}},\ Q),\Theta) &\triangleq \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/a_{\mathsf{S}}]\,Q),[b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Theta\\
\\
[b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\cdot) &\triangleq (\cdot)\\
[b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\mathsf{proc}(c_{\mathsf{S}},\ Q),\Lambda) &\triangleq \mathsf{proc}(c_{\mathsf{S}},\ [b_{\mathsf{S}}/a_{\mathsf{S}}]\,Q),[b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Lambda \qquad \textit{(where } a_{\mathsf{S}} \neq c_{\mathsf{S}}\textit{)}\\
[b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\mathsf{unavail}(c_{\mathsf{S}}),\Lambda) &\triangleq \mathsf{unavail}(c_{\mathsf{S}}),[b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Lambda
\end{aligned}$$

For a well-typed configuration $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/a_{\mathsf{L}}]\,Q),[b_{\mathsf{L}}/a_{\mathsf{L}}]\,\Theta :: \Delta$ *where* a_{L} *occurs in* Q*, it follows by induction on the configuration typing that* $[b_{\mathsf{L}}/a_{\mathsf{L}}]\,\Theta = \Theta$*. In order for* a_{L} *to be used in* Q *it must be offered by the sub-configuration* Θ *and hence cannot be used by any process in* Θ*. Both side-conditions are easily met in all cases Definition 1 is used.*

Definition 2 (Concretization). *A concretization is a partial order on structural contexts* Γ *and is inductively defined by the following rules, relying on the partial order* $\bot \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}} \leq \top$*, for any* C_{L}*:*

$$\frac{}{\Gamma \lhd (\cdot)}\ (\text{T-}\lhd_1) \qquad \frac{\hat{A} \leq \hat{B} \qquad \Gamma' \lhd \Gamma}{\Gamma', x_{\mathsf{S}} : \hat{A} \lhd \Gamma, x_{\mathsf{S}} : \hat{B}}\ (\text{T-}\lhd_2)$$

Definition 3 (Poised Process and Configuration). *A* $\mathsf{proc}(a,\ P_a)$ *is poised if it is communicating along its providing channel. The poised processes in* $\mathsf{SILL_S}$ *are:*

Receiving	*Sending*
$\mathsf{proc}(a_{\mathsf{L}},\ y \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_y)$	$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b\ ;P)$
	$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}})$
$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P})$	$\mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;P)$
$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;P_{x_{\mathsf{L}}})$	$\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ;P_{x_{\mathsf{S}}})$

A configuration Θ *is poised if and only if all* $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}) \in \Theta$ *are poised. A configuration* Λ *is poised if and only if all* $\mathsf{proc}(a_{\mathsf{S}},\ P_{a_{\mathsf{S}}}) \in \Lambda$ *are poised.*

Definition 4 (Blocked Process). *A process is* blocked along a_{S} *if it has the form* $\mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;Q_{x_{\mathsf{L}}})$*.*

D.1.2 Lemmas and Corollaries

Lemma 1 (Process Term Substitution). *Given the partial order* $\bot \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}} \leq \top$*, for any* C_{L}*, the following substitutions are type-preserving and thus admissible:*

1. *If* $\Gamma;\Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$*, then, for any fresh* $y_{\mathsf{L}} : A_{\mathsf{L}}$*,* $\Gamma;\Delta \vdash_\Sigma [y_{\mathsf{L}}/x_{\mathsf{L}}]\,P :: (y_{\mathsf{L}} : A_{\mathsf{L}})$*.*

2. *If* $\Gamma; \Delta, y_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$*, then, for any fresh* $z_{\mathsf{L}} : B_{\mathsf{L}}$, $\Gamma; \Delta, z_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{L}}/y_{\mathsf{L}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$.
3. *If* $\Gamma, y_{\mathsf{S}} : \hat{B}; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$*, then, for any fresh* $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$, $\Gamma, z_{\mathsf{S}} : \hat{C}; \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$.
4. *If* $\Gamma, y_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{C}; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$*, then* $\Gamma, y_{\mathsf{S}} : \hat{C}; \Delta \vdash_\Sigma [y_{\mathsf{S}}/y'_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$.
5. *If* $\Gamma \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$*, then, for any fresh* $y_{\mathsf{S}} : A_{\mathsf{S}}$, $\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [y_{\mathsf{S}}/x_{\mathsf{S}}]\, P :: (y_{\mathsf{S}} : A_{\mathsf{S}})$.
6. *If* $\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}}, x'_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma P :: (x'_{\mathsf{S}} : A_{\mathsf{S}})$*, then* $\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [x_{\mathsf{S}}/x'_{\mathsf{S}}]\, P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$.
7. *If* $\Gamma, y_{\mathsf{S}} : \hat{B} \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$*, then, for any fresh* $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$, $\Gamma, z_{\mathsf{S}} : \hat{C} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$.
8. *If* $\Gamma, y_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{C} \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$*, then* $\Gamma, y_{\mathsf{S}} : \hat{C} \vdash_\Sigma [y_{\mathsf{S}}/y'_{\mathsf{S}}]\, P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$.

Proof. We prove each case in turn:

Lemma 1-1: By α-equivalence.

Lemma 1-2: By α-equivalence.

Lemma 1-3 and 1-7: By simultaneous induction on $\Gamma, y_{\mathsf{S}} : \hat{B}; \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ and $\Gamma, y_{\mathsf{S}} : \hat{B} \vdash_\Sigma P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$:

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{L}}\, y_{\mathsf{L}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{L}}\, y_{\mathsf{L}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (since y_{S} does not occur in the process term)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$
- $\Gamma, y_{\mathsf{S}} : \hat{B} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{S}}\, y_{\mathsf{S}} :: (x_{\mathsf{S}} : B_{\mathsf{S}})$ and $\hat{B} \le B_{\mathsf{S}}$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{S}}\, z_{\mathsf{S}} :: (x_{\mathsf{S}} : B_{\mathsf{S}})$ (by (T-Id$_{\mathsf{S}}$) and since $\hat{C} \le B_{\mathsf{S}}$ by transitivity of $\le$)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$

- $\Gamma, x'_{\mathsf{S}} : \hat{A}, y_{\mathsf{S}} : \hat{B} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{S}}\, x'_{\mathsf{S}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ and $\hat{A} \le A_{\mathsf{S}}$ (this case)
  $\Gamma, x'_{\mathsf{S}} : \hat{A}, z_{\mathsf{S}} : \hat{C} \vdash_\Sigma \mathsf{fwd}\ x_{\mathsf{S}}\, x'_{\mathsf{S}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (since y_{S} does not occur in the process term)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta, \Delta' \vdash_\Sigma x'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}, y_{\mathsf{S}}\ ;\ Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $(x''_{\mathsf{L}} : A'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}' : B_{\mathsf{L}}}, \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}}, y'_{\mathsf{S}} : B_{\mathsf{S}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}'}, \overline{w_{\mathsf{S}}'}, y'_{\mathsf{S}}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \le \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ and $\Delta = \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta', x'_{\mathsf{L}} : A'_{\mathsf{L}} \vdash_\Sigma Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta', x'_{\mathsf{L}} : A'_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta, \Delta' \vdash_\Sigma x'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}, z_{\mathsf{S}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Spawn$_{\mathsf{LL}}$))

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta, \Delta' \vdash_\Sigma x'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}\ ;\ Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  where $y_{\mathsf{S}} \notin \overline{w_{\mathsf{S}}}$
  $(x''_{\mathsf{L}} : A'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}' : B_{\mathsf{L}}}, \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}} : B_{\mathsf{S}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}'}, \overline{w_{\mathsf{S}}'}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \le \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ and $\Delta = \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta', x'_{\mathsf{L}} : A'_{\mathsf{L}} \vdash_\Sigma Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta', x'_{\mathsf{L}} : A'_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta, \Delta' \vdash_\Sigma x'_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}}}, \overline{w_{\mathsf{S}}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Spawn$_{\mathsf{LL}}$))

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}, y_{\mathsf{S}}\ ;\ Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $(x''_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}}, y'_{\mathsf{S}} : B_{\mathsf{S}} = P_{x'_{\mathsf{S}}, \overline{w_{\mathsf{S}}'}, y'_{\mathsf{S}}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \le \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B}, x'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C}, x'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \le \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}, z_{\mathsf{S}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Spawn$_{\mathsf{LS}}$))

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}\ ;\ Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  where $y_{\mathsf{S}} \notin \overline{w_{\mathsf{S}}}$
  $(x''_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{w_{\mathsf{S}}'}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B}, x'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C}, x'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\textsc{Spawn}_{\mathsf{LS}}$))

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B} \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}, y_{\mathsf{S}}\ ;\ Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (this case)
  $(x''_{\mathsf{S}} : A'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}}, y'_{\mathsf{S}} : B_{\mathsf{S}} = P_{x'_{\mathsf{S}}, \overline{w_{\mathsf{S}}'}, y'_{\mathsf{S}}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B}, x'_{\mathsf{S}} : A'_{\mathsf{S}} \vdash_{\Sigma} Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C}, x'_{\mathsf{S}} : A'_{\mathsf{S}} \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C} \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}, z_{\mathsf{S}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-$\textsc{Spawn}_{\mathsf{SS}}$))

- $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B} \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}\ ;\ Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (this case)
  where $y_{\mathsf{S}} \notin \overline{w_{\mathsf{S}}}$
  $(x''_{\mathsf{S}} : A'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}' : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{w_{\mathsf{S}}'}}) \in \Sigma$ (this case)
  $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$ and $\Gamma = \overline{w_{\mathsf{S}} : \hat{B}}$ (this case)
  $\Gamma, \Gamma', y_{\mathsf{S}} : \hat{B}, x'_{\mathsf{S}} : A'_{\mathsf{S}} \vdash_{\Sigma} Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (this case)
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C}, x'_{\mathsf{S}} : A'_{\mathsf{S}} \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, \Gamma', z_{\mathsf{S}} : \hat{C} \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{w_{\mathsf{S}}}\ ;\ [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-$\textsc{Spawn}_{\mathsf{SS}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_{\Sigma} x'_{\mathsf{L}} \leftarrow \mathsf{acquire}\, y_{\mathsf{S}}\, ; Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\hat{B} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}}$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_{\Sigma} Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\hat{C} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}}$ (by transitivity of $\leq$)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_{\Sigma} x'_{\mathsf{L}} \leftarrow \mathsf{acquire}\, z_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\uparrow_{\mathsf{L}}^{\mathsf{S}}$L))

- $\Gamma, y_{\mathsf{S}} : \hat{B}, y'_{\mathsf{S}} : \hat{D};\ \Delta \vdash_{\Sigma} x'_{\mathsf{L}} \leftarrow \mathsf{acquire}\, y'_{\mathsf{S}}\, ; Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\hat{D} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}}$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B}, y'_{\mathsf{S}} : \hat{D};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_{\Sigma} Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{D};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{D};\ \Delta \vdash_{\Sigma} x'_{\mathsf{L}} \leftarrow \mathsf{acquire}\, y'_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x'_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\uparrow_{\mathsf{L}}^{\mathsf{S}}$L))

- $\Gamma, y_{\mathsf{S}} : \hat{B} \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\, x_{\mathsf{S}}\, ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \cdot \vdash_{\Sigma} [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C} \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\, x_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$ (by (T-$\uparrow_{\mathsf{L}}^{\mathsf{S}}$R))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_{\Sigma} x'_{\mathsf{S}} \leftarrow \mathsf{release}\, x'_{\mathsf{L}}\, ; Q_{x_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)

$\Gamma, y_{\mathsf{S}} : \hat{B}, x_{\mathsf{S}}' : A_{\mathsf{S}};\ \Delta \vdash_\Sigma Q_{x_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
$\Gamma, z_{\mathsf{S}} : \hat{C}, x_{\mathsf{S}}' : A_{\mathsf{S}};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
$\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x_{\mathsf{L}}' : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_\Sigma x_{\mathsf{S}}' \leftarrow \mathsf{release}\, x_{\mathsf{L}}' \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\downarrow_{\mathsf{L}}^{\mathsf{S}}$L))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \cdot \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{detach}\, x_{\mathsf{L}} \,; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B} \vdash_\Sigma P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \cdot \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{detach}\, x_{\mathsf{L}} \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})$ (by (T-$\downarrow_{\mathsf{L}}^{\mathsf{S}}$R))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, y_{\mathsf{L}} : \mathbf{1} \vdash_\Sigma \mathsf{wait}\, y_{\mathsf{L}} \,; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, y_{\mathsf{L}} : \mathbf{1} \vdash_\Sigma \mathsf{wait}\, y_{\mathsf{L}} \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\mathbf{1}_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \cdot \vdash_\Sigma \mathsf{close}\, x_{\mathsf{L}} :: (x_{\mathsf{L}} : \mathbf{1})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \cdot \vdash_\Sigma \mathsf{close}\, x_{\mathsf{L}} :: (x_{\mathsf{L}} : \mathbf{1})$ (since y_{S} does not occur in the process term)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x_{\mathsf{L}}' : B_{\mathsf{L}} \otimes C_{\mathsf{L}} \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}' \,; Q_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x_{\mathsf{L}}' : C_{\mathsf{L}}, y_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x_{\mathsf{L}}' : C_{\mathsf{L}}, y_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x_{\mathsf{L}}' : B_{\mathsf{L}} \otimes C_{\mathsf{L}} \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}' \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\otimes_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{L}} \,; P :: (x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{L}} \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})$ (by (T-$\otimes_{\mathrm{R}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x_{\mathsf{L}}' : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma y_{\mathsf{S}}' \leftarrow \mathsf{recv}\, x_{\mathsf{L}}' \,; Q_{y_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B}, y_{\mathsf{S}}' : A_{\mathsf{S}};\ \Delta, x_{\mathsf{L}}' : B_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y_{\mathsf{S}}' : A_{\mathsf{S}};\ \Delta, x_{\mathsf{L}}' : B_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x_{\mathsf{L}}' : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma y_{\mathsf{S}}' \leftarrow \mathsf{recv}\, x_{\mathsf{L}}' \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}'} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\exists_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{S}} \,; P :: (x_{\mathsf{L}} : (\exists B_{\mathsf{S}}.A_{\mathsf{L}}))$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ and $\hat{B} \leq B_{\mathsf{S}}$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\hat{C} \leq B_{\mathsf{S}}$ (by transitivity of $\leq$)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, z_{\mathsf{S}} \,; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : (\exists B_{\mathsf{S}}.A_{\mathsf{L}}))$ (by (T-$\exists_{\mathrm{R}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B}, y_{\mathsf{S}}' : \hat{A};\ \Delta \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, y_{\mathsf{S}}' \,; P :: (x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}))$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B}, y_{\mathsf{S}}' : \hat{A};\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ and $\hat{A} \leq A_{\mathsf{S}}$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y_{\mathsf{S}}' : \hat{A};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$

$\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{A};\ \Delta \vdash_\Sigma \mathsf{send}\, x_{\mathsf{L}}\, y'_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : (\exists A_{\mathsf{S}}.B_{\mathsf{L}}))$ (by (T-$\exists_{\mathrm{R}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, y_{\mathsf{L}}\, ; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, y_{\mathsf{L}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\multimap_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}\, ; P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{y_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}})$ (by (T-$\multimap_{\mathrm{R}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : (\Pi B_{\mathsf{S}}.C_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, y_{\mathsf{S}}\, ; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ and $\hat{B} \leq B_{\mathsf{S}}$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : C_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\hat{C} \leq B_{\mathsf{S}}$ (by transitivity of $\leq$)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : (\Pi B_{\mathsf{S}}.C_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, z_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Π_{L}))

- $\Gamma, y_{\mathsf{S}} : \hat{B}, y'_{\mathsf{S}} : \hat{A};\ \Delta, x'_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, y'_{\mathsf{S}}\, ; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B}, y'_{\mathsf{S}} : \hat{A};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ and $\hat{A} \leq A_{\mathsf{S}}$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{A};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : \hat{A};\ \Delta, x'_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\, x'_{\mathsf{L}}\, y'_{\mathsf{S}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Π_{L}))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma y'_{\mathsf{S}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}\, ; P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}))$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B}, y'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C}, y'_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : B_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma y'_{\mathsf{S}} \leftarrow \mathsf{recv}\, x_{\mathsf{L}}\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_{y_{\mathsf{S}}} :: (x_{\mathsf{L}} : (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}))$ (by (T-Π_{R}))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : \oplus\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma \mathsf{case}\, x'_{\mathsf{L}}\, \mathsf{of}\, \overline{l \Rightarrow Q} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $(\forall i)$ $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}_i} \vdash_\Sigma Q_i :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $(\forall i)$ $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}_i} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_i :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : \oplus\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma \mathsf{case}\, x'_{\mathsf{L}}\, \mathsf{of}\, \overline{l \Rightarrow [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\oplus_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma x_{\mathsf{L}}.l_h\, ; P :: (x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta \vdash_\Sigma P :: (x_{\mathsf{L}} : A_{\mathsf{L}\,h})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : A_{\mathsf{L}\,h})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta \vdash_\Sigma x_{\mathsf{L}}.l_h\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P :: (x_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})$ (by (T-$\oplus_{\mathrm{R}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : \&\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma x'_{\mathsf{L}}.l_h\, ; Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, y_{\mathsf{S}} : \hat{B};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}\,h} \vdash_\Sigma Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : B_{\mathsf{L}\,h} \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\ \Delta, x'_{\mathsf{L}} : \&\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma x'_{\mathsf{L}}.l_h\, ; [z_{\mathsf{S}}/y_{\mathsf{S}}]\, Q :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-$\&_{\mathrm{L}}$))

- $\Gamma, y_{\mathsf{S}} : \hat{B};\, \Delta \vdash_\Sigma \mathsf{case}\ x_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P} :: (x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})$ (this case)
  $(\forall i)\ \ \Gamma, y_{\mathsf{S}} : \hat{B};\, \Delta \vdash_\Sigma P_i :: (x_{\mathsf{L}} : A_{\mathsf{L}_i})$ (this case)
  $(\forall i)\ \ \Gamma, z_{\mathsf{S}} : \hat{C};\, \Delta \vdash_\Sigma [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P_i :: (x_{\mathsf{L}} : A_{\mathsf{L}_i})$ (by I.H.)
  for some fresh $z_{\mathsf{S}} : \hat{C}$ *such that* $\hat{C} \leq \hat{B}$
  $\Gamma, z_{\mathsf{S}} : \hat{C};\, \Delta \vdash_\Sigma \mathsf{case}\ x_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow [z_{\mathsf{S}}/y_{\mathsf{S}}]\, P} :: (x_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})$ ((T-$\&_{\mathrm{R}}$))

Lemma 1-4: By contraction.

Lemma 1-5: By α-equivalence.

Lemma 1-6: By contraction.

Lemma 1-8: By contraction.

□

Lemma 2 (Configuration Substitution)**.** *Writing $Q\langle a\rangle$ for a process term Q with an occurrence of a channel a and given the partial order $\bot \leq {\uparrow}^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{L}} \leq \top$, for any C_{L}, the following substitutions are type-preserving and thus admissible:*

1. *If $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta :: \Delta$ and $\Gamma \vDash_\Sigma \Theta :: (\Delta, \Delta_1, a_{\mathsf{L}} : A_{\mathsf{L}})$, then, for any Θ' such that $\Gamma \vDash_\Sigma \Theta' :: (\Delta, \Delta_1, b_{\mathsf{L}} : A_{\mathsf{L}})$, $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/a_{\mathsf{L}}]\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta' :: \Delta$.*
2. *If $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Theta :: \Delta$ and $\hat{B} \leq \hat{A}$, then $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\, \Theta :: \Delta$.*
3. *If $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda :: \Gamma_1$ and $\mathsf{proc}(a_{\mathsf{S}},\ \_) \notin \Lambda$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda$ and $\hat{B} \leq \hat{A}$, then $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\, \Lambda :: \Gamma_1$.*

Proof. We prove each case in turn:

1. By induction on $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta :: \Delta$:
   - $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta :: \Delta$ and $\Gamma \vDash_\Sigma \Theta :: (\Delta, \Delta_1, a_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
     $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (this case)
     $\Gamma; \Delta_1, a_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q\langle a_{\mathsf{L}}\rangle :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (this case)
     $\Gamma; \Delta_1, b_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{L}}/a_{\mathsf{L}}]\ Q\langle a_{\mathsf{L}}\rangle :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-2)
     $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/a_{\mathsf{L}}]\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta' :: \Delta$ (by (T-Ω_2))
     for some Θ' *such that* $\Gamma \vDash_\Sigma \Theta' :: (\Delta, \Delta_1, b_{\mathsf{L}} : A_{\mathsf{L}})$
2. By induction on $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Theta :: \Delta$:
   - $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ and $\hat{B} \leq \hat{A}$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ (by Definition 1)
   - $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C} \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta :: (\Delta, a_{\mathsf{L}} : A_{\mathsf{L}})$ and $\hat{C} \leq \hat{B}$ (this case)
     $(a_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\vdash_\Sigma (A_{\mathsf{L}}, \hat{B})\ \mathsf{esync}$ (this case)
     $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C};\, \Delta' \vdash_\Sigma P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
     $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C} \vDash_\Sigma \Theta : \Delta, \Delta'$ (this case)
     $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C} \vDash_\Sigma [c_{\mathsf{S}}/b_{\mathsf{S}}]\, \Theta : \Delta, \Delta'$ (by I.H.)
     $\Gamma, c'_{\mathsf{S}} : \hat{C}, c_{\mathsf{S}} : \hat{C};\, \Delta' \vdash_\Sigma [c'_{\mathsf{S}}/b_{\mathsf{S}}]\, P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-3)
     for some fresh $c'_{\mathsf{S}} : \hat{C}$
     $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C};\, \Delta' \vdash_\Sigma [c_{\mathsf{S}}/c'_{\mathsf{S}}]\, ([c'_{\mathsf{S}}/b_{\mathsf{S}}]\, P_{a_{\mathsf{L}}}) :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-4 and weakening)
     $\Gamma, b_{\mathsf{S}} : \hat{B}, c_{\mathsf{S}} : \hat{C} \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [c_{\mathsf{S}}/b_{\mathsf{S}}]\, P_{a_{\mathsf{L}}}),\ [c_{\mathsf{S}}/b_{\mathsf{S}}]\, \Theta :: (\Delta, a_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Θ_2))
3. By induction on $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda :: \Gamma_1$:
   - $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ and $\mathsf{proc}(a_{\mathsf{S}},\ \_) \notin (\cdot)$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin (\cdot)$ and $\hat{B} \leq \hat{A}$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ (by Definition 1)
   - $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \mathsf{proc}(c_{\mathsf{S}},\ P_{c_{\mathsf{S}}}) :: (c_{\mathsf{S}} : {\uparrow}^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{L}})$ and $\hat{B} \leq \hat{A}$ and $a_{\mathsf{S}} \neq c_{\mathsf{S}}$ (this case)
     $\vdash_\Sigma ({\uparrow}^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{L}}, \top)\ \mathsf{esync}$ (this case)

$\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vdash_\Sigma P_{a_{\mathsf{S}}} :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (this case)
$\Gamma, b'_{\mathsf{S}} : \hat{B}, b_{\mathsf{S}} : \hat{B} \vdash_\Sigma [b'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{a_{\mathsf{S}}} :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (by Lemma 1-7)
for some fresh $b'_{\mathsf{S}} : \hat{B}$
$\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vdash_\Sigma [b_{\mathsf{S}}/b'_{\mathsf{S}}]\, ([b'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{a_{\mathsf{S}}}) :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (by Lemma 1-8 and weakening)
$\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vdash_\Sigma \mathsf{proc}(c_{\mathsf{S}},\, [b_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{a_{\mathsf{S}}}) :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (by (T-Λ_2))

- $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \mathsf{unavail}(c_{\mathsf{S}}) :: (c_{\mathsf{S}} : \hat{C})$ and $\hat{B} \leq \hat{A}$ and $a_{\mathsf{S}} \neq c_{\mathsf{S}}$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \mathsf{unavail}(c_{\mathsf{S}}) :: (c_{\mathsf{S}} : \hat{C})$ (by Definition 1)
- $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_1, \Lambda_2 :: \Gamma_1, \Gamma_2, b_{\mathsf{S}} : \hat{B}$ and $\mathsf{proc}(a_{\mathsf{S}},\, \_) \notin \Lambda_1, \Lambda_2$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda_1, \Lambda_2$ (this case)
  for some Λ_1, Λ_2, Γ_1, *and* Γ_2
  $\hat{B} \leq \hat{A}$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_1 :: \Gamma_1, b_{\mathsf{S}} : \hat{B}$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_2 :: \Gamma_2$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\, \Lambda_1 :: \Gamma_1, b_{\mathsf{S}} : \hat{B}$ (by I.H.)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\, \Lambda_2 :: \Gamma_2$ (by I.H.)
  $\Gamma, a_{\mathsf{S}} : \hat{A}, b_{\mathsf{S}} : \hat{B} \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\, (\Lambda_1, \Lambda_2) :: \Gamma_1, \Gamma_2, b_{\mathsf{S}} : \hat{B}$ ((T-Λ_4))

□

Lemma 3 (Update of Γ)**.** *Given the partial order* $\bot \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}} \leq \top$, *for any* C_{L}, *the following substitutions are type-preserving and thus admissible:*

1. *If* $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Theta :: \Delta$, *then, for any* C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$ *and* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{B})$ esync, $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Theta :: \Delta$.
2. *If* $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Lambda :: \Gamma_1$ *with* $\mathsf{proc}(a_{\mathsf{S}}, P) \notin \Lambda$ *and* $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda$, *then, for any* C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$ *and* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{B})$ esync, $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda :: \Gamma_1$.

Proof. We prove each case in turn:

1. By induction on $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Theta :: \Delta$:
   - $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma (\cdot) :: (\cdot)$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ (since a_{S} does not occur in the configuration)
     for some C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$
   - $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta :: (\Delta, c_{\mathsf{L}} : D_{\mathsf{L}})$ (this case)
     $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (D_{\mathsf{L}}, \hat{D})$ esync (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{A};\, \Delta' \vdash_\Sigma P_{c_{\mathsf{L}}} :: (c_{\mathsf{L}} : D_{\mathsf{L}})$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Theta : \Delta, \Delta'$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Theta : \Delta, \Delta'$ (by I.H.)
     for some C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$
     $\Gamma, a'_{\mathsf{S}} : \hat{B};\, \Delta' \vdash_\Sigma [a'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{c_{\mathsf{L}}} :: (c_{\mathsf{L}} : D_{\mathsf{L}})$ (by Lemma 1-3)
     for some fresh $a'_{\mathsf{S}} : \hat{B}$
     $\Gamma, a_{\mathsf{S}} : \hat{B};\, \Delta' \vdash_\Sigma [a_{\mathsf{S}}/a'_{\mathsf{S}}]\, ([a'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{c_{\mathsf{L}}}) :: (c_{\mathsf{L}} : D_{\mathsf{L}})$ (by Lemma 1-4 and weakening)
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta :: (\Delta, c_{\mathsf{L}} : D_{\mathsf{L}})$ (by (T-Θ_2))
2. By induction on $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Lambda :: \Gamma_1$:
   - $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma (\cdot) :: (\cdot)$ and $\mathsf{proc}(a_{\mathsf{S}},\, \_) \notin (\cdot)$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin (\cdot)$ (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma (\cdot) :: (\cdot)$ (since a_{S} does not occur in the configuration)
     for some C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$
   - $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \mathsf{proc}(c_{\mathsf{S}},\, P_{c_{\mathsf{S}}}) :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}})$ and $a_{\mathsf{S}} \neq c_{\mathsf{S}}$ (this case)
     $\vdash_\Sigma (\uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}}, \top)$ esync (this case)
     $\Gamma, a_{\mathsf{S}} : \hat{A} \vdash_\Sigma P_{a_{\mathsf{S}}} :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}})$ (this case)
     $\Gamma, b'_{\mathsf{S}} : \hat{B} \vdash_\Sigma [b'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{a_{\mathsf{S}}} :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}})$ (by Lemma 1-7)
     for some fresh $b'_{\mathsf{S}} : \hat{B}$ *and* C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and* $\hat{B} \leq \hat{A}$
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vdash_\Sigma [a_{\mathsf{S}}/b'_{\mathsf{S}}]\, ([b'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{a_{\mathsf{S}}}) :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}})$ (by Lemma 1-8 and weakening)
     $\Gamma, a_{\mathsf{S}} : \hat{B} \vdash_\Sigma \mathsf{proc}(c_{\mathsf{S}},\, P_{a_{\mathsf{S}}}) :: (c_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} D_{\mathsf{L}})$ (by (T-Λ_2))

- $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \mathsf{unavail}(c_{\mathsf{S}}) :: (c_{\mathsf{S}} : \hat{C})$ and $a_{\mathsf{S}} \neq c_{\mathsf{S}}$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \mathsf{unavail}(c_{\mathsf{S}}) :: (c_{\mathsf{S}} : \hat{C})$ (since a_{S} does not occur in the configuration)
- $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Lambda_1, \Lambda_2 :: \Gamma_1, \Gamma_2$ and $\mathsf{proc}(a_{\mathsf{S}}, \_) \notin \Lambda_1, \Lambda_2$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda_1, \Lambda_2$ (this case)
  for some Λ_1, Λ_2, Γ_1, *and* Γ_2
  $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Lambda_1 :: \Gamma_1$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{A} \vDash_\Sigma \Lambda_2 :: \Gamma_2$ (this case)
  $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_1 :: \Gamma_1$ (by I.H.)
  for some C_{L} *such that* $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})$ esync *and for any* $\hat{B}$ *such that* $\hat{B} \leq \hat{A}$
  $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_2 :: \Gamma_2$ (by I.H.)
  $\Gamma, a_{\mathsf{S}} : \hat{B} \vDash_\Sigma \Lambda_1, \Lambda_2 :: \Gamma_1, \Gamma_2$ ((T-Λ_4))

□

Lemma 4 (Equi-Synchronizing and $\bot$)**.** *If* $\vdash_\Sigma (C_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})$ esync *and* $\vdash_\Sigma (C_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}})$ esync*, for any* C_{L}*,* $\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$*,* $\uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}$ *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$*, then* $\vdash_\Sigma (C_{\mathsf{L}}, \bot)$ esync*.*

Proof. For the presentation of the proof, we switch to a set-based formulation of esync to allow for a more natural formulation of the coinductive proof. The assertion $\vdash_\Sigma (D, \hat{D})$ esync then is expressed as $(D, \hat{D}) \in \mathsf{esync}$. Given the monotone generating function $\mathcal{F}$ arising from the rules defined in Figure 16, we obtain that $\mathsf{esync} \subseteq \mathcal{F}(\mathsf{esync})$, since esync is $\mathcal{F}$-consistent.

Next, we define the set esync' as

$$\mathsf{esync}' \triangleq \mathsf{esync} \cup \mathsf{esync}_\bot$$

where $\mathsf{esync}_\bot$ is defined as:

$$\mathsf{esync}_\bot \triangleq \{(C_{\mathsf{L}}, \bot) \mid \exists A_{\mathsf{L}}, B_{\mathsf{L}} \,.\, (C_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync} \wedge (C_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync} \wedge A_{\mathsf{L}} \neq B_{\mathsf{L}}\}$$

To prove Lemma 4 it suffices to show that esync' is $\mathcal{F}$-consistent, i.e., that $\mathsf{esync}' \subseteq \mathcal{F}(\mathsf{esync}')$. Thus, we have to show the following two cases:

(i) $\mathsf{esync} \subseteq \mathcal{F}(\mathsf{esync}')$ and
(ii) $\mathsf{esync}_\bot \subseteq \mathcal{F}(\mathsf{esync}')$

We first show *(i)* and then *(ii)*:

(i) $\mathsf{esync} \subseteq \mathsf{esync}'$ (by definition of esync')
$\mathcal{F}(\mathsf{esync}) \subseteq \mathcal{F}(\mathsf{esync}')$ (by monotonicity of $\mathcal{F}$)
$\mathsf{esync} \subseteq \mathcal{F}(\mathsf{esync})$ (since esync is $\mathcal{F}$-consistent)
$\mathsf{esync} \subseteq \mathcal{F}(\mathsf{esync}')$ (by transitivity of $\subseteq$)

(ii) We consider each syntactic form of C_{L} in turn:

- $(\oplus\{\overline{l : C'_{\mathsf{L}}}\}, \bot) \in \mathsf{esync}_\bot$ (this case)
  $(\oplus\{\overline{l : C'_{\mathsf{L}}}\}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\oplus\{\overline{l : C'_{\mathsf{L}}}\}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\forall i)$ $(C'_{\mathsf{L}_i}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-Esync$_\oplus$))
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \bot) \in \mathsf{esync}_\bot$ (by definition of $\mathsf{esync}_\bot$)
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_\bot \subseteq \mathsf{esync}'$)
  $(\oplus\{\overline{l : C'_{\mathsf{L}}}\}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-Esync$_\oplus$))

- $(\&\{\overline{l : C'_{\mathsf{L}}}\}, \bot) \in \mathsf{esync}_\bot$ (this case)
  $(\&\{\overline{l : C'_{\mathsf{L}}}\}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\&\{\overline{l : C'_{\mathsf{L}}}\}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\forall i)$ $(C'_{\mathsf{L}_i}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-Esync$_\&$))
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \bot) \in \mathsf{esync}_\bot$ (by definition of $\mathsf{esync}_\bot$)
  $(\forall i)$ $(C'_{\mathsf{L}_i}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_\bot \subseteq \mathsf{esync}'$)
  $(\&\{\overline{l : C'_{\mathsf{L}}}\}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-Esync$_\&$))

- $(C'_{\mathsf{L}} \otimes C''_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  $(C'_{\mathsf{L}} \otimes C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C'_{\mathsf{L}} \otimes C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-ESYNC$_\otimes$))
  $(C''_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (by definition of $\mathsf{esync}_{\bot}$)
  $(C''_{\mathsf{L}}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_{\bot} \subseteq \mathsf{esync}'$)
  $(C'_{\mathsf{L}} \otimes C''_{\mathsf{L}}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-ESYNC$_\otimes$))

- $(C'_{\mathsf{L}} \multimap C''_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  $(C'_{\mathsf{L}} \multimap C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C'_{\mathsf{L}} \multimap C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C''_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-ESYNC$_\multimap$))
  $(C''_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (by definition of $\mathsf{esync}_{\bot}$)
  $(C''_{\mathsf{L}}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_{\bot} \subseteq \mathsf{esync}'$)
  $(C'_{\mathsf{L}} \multimap C''_{\mathsf{L}}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-ESYNC$_\multimap$))

- $(\exists C_{\mathsf{S}}.C'_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  $(\exists C_{\mathsf{S}}.C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\exists C_{\mathsf{S}}.C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-ESYNC$_\exists$))
  $(C'_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (by definition of $\mathsf{esync}_{\bot}$)
  $(C'_{\mathsf{L}}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_{\bot} \subseteq \mathsf{esync}'$)
  $(\exists C_{\mathsf{S}}.C'_{\mathsf{L}}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-ESYNC$_\exists$))

- $(\Pi C_{\mathsf{S}}.C'_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  $(\Pi C_{\mathsf{S}}.C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\Pi C_{\mathsf{S}}.C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  $(C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(C'_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (by inversion on (T-ESYNC$_\exists$))
  $(C'_{\mathsf{L}}, \bot) \in \mathsf{esync}_{\bot}$ (by definition of $\mathsf{esync}_{\bot}$)
  $(C'_{\mathsf{L}}, \bot) \in \mathsf{esync}'$ (since $\mathsf{esync}_{\bot} \subseteq \mathsf{esync}'$)
  $(\Pi C_{\mathsf{S}}.C'_{\mathsf{L}}, \bot) \in \mathcal{F}(\mathsf{esync}')$ (by (T-ESYNC$_\exists$))

- $(\mathbf{1}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  Nothing to show.

- $(\downarrow^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{S}}, \bot) \in \mathsf{esync}_{\bot}$ (this case)
  $(\downarrow^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{S}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \mathsf{esync}$ and $(\downarrow^{\mathsf{S}}_{\mathsf{L}} C_{\mathsf{S}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \mathsf{esync}$ (this case)
  for some A_{L} *and* B_{L} *such that* $A_{\mathsf{L}} \neq B_{\mathsf{L}}$
  Contradiction. (since $A_{\mathsf{L}} = B_{\mathsf{L}}$ or $A_{\mathsf{L}} = B_{\mathsf{L}} = \top$ according to (T-ESYNC$_{\downarrow^{\mathsf{S}}_{\mathsf{L}}}$-1) or (T-ESYNC$_{\downarrow^{\mathsf{S}}_{\mathsf{L}}}$-2), resp.)

□

Corollary 1 (Balance between Λ and Θ)**.** *If the configuration* $\Lambda; \Theta$ *is well-formed, then, for any* $\mathsf{proc}(a_{\mathsf{S}}, \_)$ *in* Λ*, there does not exist a* $\mathsf{proc}(a_{\mathsf{L}}, \_)$ *in* Θ*.*

Proof. By well-formedness of $\Lambda; \Theta$ (see Section B.3) it follows that $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda$ and that $\forall a.\mathsf{proc}(a_{\mathsf{L}}, \_) \in \Theta \implies \mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$. Then, by the contrapositive, it follows that $\mathsf{proc}(a_{\mathsf{L}}, \_) \notin \Theta$. □

Lemma 5 (Permutation of Θ)**.** *Writing* $Q\langle a_{\mathsf{L}}\rangle$ *for a process term* Q *with an occurrence of a linear channel* a_{L}*, the following permutation is admissible:*

If $\Gamma \vDash_{\Sigma} \Theta_1, \mathsf{proc}(c_{\mathsf{L}}, Q\langle a_{\mathsf{L}}\rangle), \Theta_2, \mathsf{proc}(a_{\mathsf{L}}, P_{a_{\mathsf{L}}}), \Theta_3 :: \Delta$*, then* $\Gamma \vDash_{\Sigma} \Theta_1, \mathsf{proc}(c_{\mathsf{L}}, Q\langle a_{\mathsf{L}}\rangle), \mathsf{proc}(a_{\mathsf{L}}, P_{a_{\mathsf{L}}}), \Theta_2, \Theta_3 :: \Delta$.

Proof. In the given typing derivation, $\mathsf{proc}(a_{\mathsf{L}}, P_{a_{\mathsf{L}}})$ provides $a_{\mathsf{L}} : A_{\mathsf{L}}$ for some A_{L} to $\Theta_1, \mathsf{proc}(c_{\mathsf{L}}, Q\langle a_{\mathsf{L}}\rangle), \Theta_2$. Well-formedness of a configuration (see Section B.3) guarantees that a_{L} is unique within a configuration, and hence there can be no other process in Θ_2 that provides a service along a_{L}. Moreover, channel a_{L} cannot be consumed by Θ_2

because channels are linear and a_{L} occurs in $\mathsf{proc}(c_{\mathsf{L}},\, Q\langle a_{\mathsf{L}}\rangle)$. Any channels used by $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}})$ will continue to be available if we move the process to the left. ☐

Lemma 6 (Truncate Θ)**.** *If $\Gamma \vDash_\Sigma \Theta_1, \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2 :: \Delta$, then there exists an A_{L} and Δ' such that $\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2 :: (\Delta', a_{\mathsf{L}} : A_{\mathsf{L}})$.*

Proof. By induction on $\Gamma \vDash_\Sigma \Theta :: \Delta$:

- $\Gamma \vDash_\Sigma (\cdot) :: (\cdot)$ (this case)
  Holds vacuously.

- $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta :: \Delta$ (this case)
  $\Gamma; \Delta_2 \vdash_\Sigma P_{c_{\mathsf{L}}} :: (c : C_{\mathsf{L}})$ (this case)
  for some Δ_2, Δ_1, and C_{L} such that $\Delta = (\Delta_1, c : C_{\mathsf{L}})$
  $\Gamma \vDash_\Sigma \Theta :: \Delta_1, \Delta_2$ (this case)
  (a) $\Theta_1 = (\cdot)$ and $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2 = \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta$ (this subcase)
  $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta :: (\Delta_1, c : C_{\mathsf{L}})$
  (b) $\Theta_1 = \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta_1'$ and $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2 = \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2$ (this subcase)
  for some Θ_1' such that $\Theta = \Theta_1', \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2$
  $\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}), \Theta_2 :: (\Delta_3, a_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  for some Δ_3 and A_{L}

☐

Lemma 7 (Linear Process Terms are Stable under Concretization)**.** *If $\Gamma_1, \Gamma_2; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ and $\Gamma_2' \lhd \Gamma_2$, then $\Gamma_1, \Gamma_2'; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$.*

Proof. By induction on $\Gamma_2' \lhd \Gamma_2$:

- $\Gamma_2' \lhd (\cdot)$ (this case)
  $\Gamma_1; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by assumption)
  $\Gamma_1, \Gamma_2'; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by weakening)

- $\Gamma_3', x_{\mathsf{S}} : \hat{A} \lhd \Gamma_3, x_{\mathsf{S}} : \hat{B}$ and $\Gamma_3' \lhd \Gamma_3$ and $\hat{A} \leq \hat{B}$ (this case)
  where $\Gamma_2 = \Gamma_3, x_{\mathsf{S}} : \hat{B}$
  $\Gamma_1, \Gamma_3, x_{\mathsf{S}} : \hat{B}; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by assumption)
  $\Gamma_1, \Gamma_3', x_{\mathsf{S}} : \hat{B}; \Delta \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by I.H.)
  $\Gamma_1, \Gamma_3', x_{\mathsf{S}}' : \hat{A}; \Delta \vdash_\Sigma [x_{\mathsf{S}}'/x_{\mathsf{S}}]\, P :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-3)
  where x_{S}' fresh
  $\Gamma_1, \Gamma_3', x_{\mathsf{S}} : \hat{A}; \Delta \vdash_\Sigma [x_{\mathsf{S}}/x_{\mathsf{S}}']\, ([x_{\mathsf{S}}'/x_{\mathsf{S}}]\, P) :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-4 and weakening)

☐

Lemma 8 (Invariant Sub-Configuration in Θ)**.** *If $\Gamma \vDash_\Sigma \Theta_1, \Theta_2 :: \Delta$ and $\Gamma \vDash_\Sigma \Theta_2 :: \Delta'$, then, for any Γ' and Θ_2' such that $\Gamma' \vDash_\Sigma \Theta_2' :: \Delta'$ and $\Gamma' \lhd \Gamma$, $\Gamma' \vDash_\Sigma \Theta_1, \Theta_2' :: \Delta$.*

Proof. By induction on $\Gamma \vDash_\Sigma \Theta :: \Delta$:

- $\Gamma \vDash_\Sigma (\cdot) :: (\cdot)$ (this case)
  Holds vacuously.

- $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta :: \Delta$ (this case)
  $\Gamma; \Delta_2 \vdash_\Sigma P_{c_{\mathsf{L}}} :: (c : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{B})\ \mathsf{esync}$ (this case)
  for some Δ_2, Δ_1, C_{L}, and $\hat{B}$ such that $\Delta = (\Delta_1, c : C_{\mathsf{L}})$
  $\Gamma \vDash_\Sigma \Theta :: \Delta_1, \Delta_2$ (this case)

(a) $\Theta_1 = (\cdot)$ and $\Theta_2 = \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta$ and $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta :: \Delta$ (this subcase)
$\Gamma' \lhd \Gamma$ (by assumption)
$\Gamma' \vDash_\Sigma \Theta_2' :: \Delta$ (by assumption)

(b) $\Theta_1 = \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta_1'$ and $\Gamma \vDash_\Sigma \Theta_2 :: \Delta'$ (this subcase)
for some Θ_1' such that $\Theta = \Theta_1', \Theta_2$
$\Gamma' \vDash_\Sigma \Theta_2 :: \Delta'$ (by assumption)
$\Gamma' \lhd \Gamma$ (by assumption)
$\Gamma' \vDash_\Sigma \Theta_1', \Theta_2' :: \Delta_1, \Delta_2$ (by I.H.)
$\Gamma'; \Delta_2 \vdash_\Sigma P_{c_{\mathsf{L}}} :: (c : C_{\mathsf{L}})$ (by Lemma 7)
$\Gamma' \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta_1', \Theta_2' :: \Delta$ (by (T-Θ_2))

□

Lemma 9 (Existence of Process for Offered Linear Channel). *If $\Gamma \vDash_\Sigma \Theta :: \Delta$, then, for all $a_{\mathsf{L}} \in \mathsf{dom}(\Delta)$, there exists exactly one $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}})$, for some P, in Θ.*

Proof. We prove Lemma 9 by induction $\Gamma \vDash_\Sigma \Theta :: \Delta$:

- $\Gamma \vDash_\Sigma (\cdot) :: (\cdot)$ (this case)
  Holds vacuously.
- $\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta :: (\Delta, a_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  $\Gamma; \Delta' \vdash_\Sigma P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (this case)
  for some Δ'
  $\Gamma \vDash_\Sigma \Theta :: (\Delta, \Delta')$ (this case)
  For all $b_{\mathsf{L}} \in \mathsf{dom}(\Delta, \Delta')$, there exists exactly one $\mathsf{proc}(b_{\mathsf{L}},\ Q_{b_{\mathsf{L}}})$, for some Q, in Θ. (by I.H.)
  There exists exactly one $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}})$ in $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta$ that offers along a_{L}.
  (by well-formedness of configuration)

□

Lemma 10 (Existence of Provider). *Writing $Q\langle a_m\rangle$ for a process term Q with an occurrence of a channel a_m, the following hold:*

1. *If $\Gamma \vDash_\Sigma \Lambda;\ \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{S}}\rangle),\ \Theta_1 :: \Gamma;\ \Delta$, then either there exists a $\mathsf{proc}(a_{\mathsf{S}},\ P_{a_{\mathsf{S}}})$ in Λ, for some P, or there exists an $\mathsf{unavail}(a_{\mathsf{S}})$ in Λ.*
2. *If $\Gamma \vDash_\Sigma \Lambda;\ \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta_1 :: \Gamma;\ \Delta$, then there exists exactly one $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}})$, for some P, in Θ_1.*

Proof. We prove each case in turn:

1. $\Gamma \vDash_\Sigma \Lambda;\ \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{S}}\rangle),\ \Theta_1 :: \Gamma;\ \Delta$ (by assumption)
   $\Gamma \vDash_\Sigma \Lambda :: \Gamma$ and $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{S}}\rangle),\ \Theta_1 :: \Delta$ (by inversion on (T-Ω))
   $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ (by the meaning of the configuration and process typing judgment)
   for some $\hat{A}$
   $\Gamma \vDash_\Sigma \Lambda_1 :: (a_{\mathsf{S}} : \hat{A})$ and $\Gamma \vDash_\Sigma \Lambda_2 :: \Gamma_2$ (by inversion on (T-Λ_4))
   for some Λ_1, Λ_2, and Γ_1 such that $\Gamma = \Gamma_2, a_{\mathsf{S}} : \hat{A}$
   Either $\Lambda_1 = \mathsf{unavail}(a_{\mathsf{S}})$ or $\Lambda_1 = \mathsf{proc}(a_{\mathsf{S}},\ P_{a_{\mathsf{S}}})$ (by inversion on (T-Λ_2) and (T-Λ_2))
   for some P
2. $\Gamma \vDash_\Sigma \Lambda;\ \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta_1 :: \Gamma;\ \Delta$ (by assumption)
   $\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q\langle a_{\mathsf{L}}\rangle),\ \Theta_1 :: \Delta$ (by inversion on (T-Ω))
   $\Gamma;\ \Delta', a_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $\Gamma \vDash_\Sigma \Theta_1 :: \Delta_1, \Delta', a_{\mathsf{L}} : A_{\mathsf{L}}$
   for some Δ', Δ_1, A_{L}, and C_{L} such that $\Delta = \Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}}$
   (by inversion on (T-Θ_2) and meaning of process typing judgment)
   There exists exactly one $\mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}})$, for some P, in Θ_1 (by Lemma 9)

□

D.2 Preservation

The preservation theorem expresses that the types of the providing linear channels are maintained along transitions and that new shared channels may be allocated and the types of existing shared channels concretized.

Theorem 3 (Preservation). *If* $\Gamma \vDash_\Sigma \Lambda; \Theta :: \Gamma; \Delta$ *and* $\Lambda; \Theta \longrightarrow \Lambda'; \Theta'$*, then* $\Gamma' \vDash_\Sigma \Lambda'; \Theta' :: \Gamma'; \Delta$*, for some* Λ'*,* Θ'*, and* Γ' *such that* $\Gamma' \lhd \Gamma$.

Proof. We prove preservation by induction on the dynamics, constructing a derivation of a well-formed and well-typed configuration $\Gamma' \vDash_\Sigma \Lambda''; \Theta'' :: \Gamma'; \Delta$, where Λ'' and Θ'' are permutations of Λ' and Θ', respectively:

Case:

$$\begin{array}{l} \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}) \\ \longrightarrow a_{\mathsf{L}} = b_{\mathsf{L}},\, a_{\mathsf{S}} = b_{\mathsf{S}} \end{array} \qquad (\text{D-ID}_{\mathsf{L}})$$

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \Theta_2,\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_3 :: \Gamma;\ \Delta$
for some Λ_1, Θ_1, Θ_2, Θ_3, *and* $\mathsf{proc}(b_{\mathsf{L}},\, P)$ (by assumption and Lemma 10-2)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \Theta_2,\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_3 :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_2,\, \Theta_3 :: \Gamma;\ \Delta$ (by Lemma 5)

$\mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_2,\, \Theta_3$ (this case)
$\longrightarrow \mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, [b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Lambda_1;\ [b_{\mathsf{S}}/a_{\mathsf{S}}, b_{\mathsf{L}}/a_{\mathsf{L}}]\Theta_1,\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, [b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\Theta_2, \Theta_3)$

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\, \mathsf{unavail}(b_{\mathsf{S}}),\, \Lambda_1 :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}) :: \Gamma_1$ and $\Gamma \vDash_\Sigma \mathsf{unavail}(b_{\mathsf{S}}) :: \Gamma_2$ and $\Gamma \vDash_\Sigma \Lambda_1 :: \Gamma_3$ (by inversion on (T-Λ_4))
for some Γ_1, Γ_2, *and* Γ_3 *such that* $\Gamma = \Gamma_1, \Gamma_2, \Gamma_3$

$\Gamma_1 = a_{\mathsf{S}} : \hat{A}$ and $\Gamma_2 = b_{\mathsf{S}} : \hat{B}$ (by inversion on (T-Λ_3))
for some $\hat{A}$ *and* $\hat{B}$

$\Gamma \vDash_\Sigma \Theta_1,\, \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_2,\, \Theta_3 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_2,\, \Theta_3 :: (\Delta_1, a_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma;\ \Delta_1' \vdash_\Sigma \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' *and since* $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$

$\Delta_1' = b_{\mathsf{L}} : C_{\mathsf{L}}$ (by inversion on (T-ID_{L}))

$\Gamma \vDash_\Sigma \mathsf{proc}(b_{\mathsf{L}},\, P),\, \Theta_2,\, \Theta_3 :: (\Delta_1, b_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma;\ \Delta_1'' \vdash_\Sigma P :: (b_{\mathsf{L}} : C_{\mathsf{L}})$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{B})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1'' *and since* $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$

$\Gamma \vDash_\Sigma \Theta_2,\, \Theta_3 :: (\Delta_1, \Delta_1'')$ (by inversion on (T-Θ_2))

Since $a_{\mathsf{S}} = \hat{A}$ and $b_{\mathsf{S}} = \hat{B}$ there are 9 combinations of 2 out of $\bot$, $\top$, and $\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$ / $\uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$, for some A_{L} and B_{L}. We consider each case, collating several cases that have the same proof:

Subcases: $\hat{A} \geq \hat{B}$

$$\begin{array}{lrclrl} a_{\mathsf{S}} : & \bot & \text{and} & b_{\mathsf{S}} : & \bot & \\ a_{\mathsf{S}} : & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : & \bot & \\ a_{\mathsf{S}} : & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : & \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} & \text{and } A_{\mathsf{L}} = B_{\mathsf{L}} \\ a_{\mathsf{S}} : & \top & \text{and} & b_{\mathsf{S}} : & \bot & \\ a_{\mathsf{S}} : & \top & \text{and} & b_{\mathsf{S}} : & \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} & \\ a_{\mathsf{S}} : & \top & \text{and} & b_{\mathsf{S}} : & \top & \end{array}$$

$\Gamma \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\Theta_2, \Theta_3) :: (\Delta_1, \Delta_1'')$ (by Lemma 2-2)

$\Gamma \vDash_\Sigma \mathsf{proc}(b_{\mathsf{L}},\, P),\, [b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\Theta_2, \Theta_3) :: (\Delta_1, b_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma [b_{\mathsf{S}}/a_{\mathsf{S}}, b_{\mathsf{L}}/a_{\mathsf{L}}]\Theta_1,\, \mathsf{proc}(b_{\mathsf{L}},\, P),\, [b_{\mathsf{S}}/a_{\mathsf{S}}]\,(\Theta_2, \Theta_3) :: \Delta$ (by Lemma 8 and Lemma 2-1 and 2-2)

$\Gamma \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma_3$ (by Lemma 2-3 since $\mathsf{proc}(a_\mathsf{S},\,\_) \notin \Lambda_1$ and $\mathsf{unavail}(a_\mathsf{S}) \notin \Lambda_1$)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma$ (by (T-Λ_4))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\; [b_\mathsf{S}/a_\mathsf{S}, b_\mathsf{L}/a_\mathsf{L}]\Theta_1,\, \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: \Gamma;\, \Delta$
(by (T-Ω) and well-formedness maintained)

Subcase: $a_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$ and $b_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}$ and $A_\mathsf{L} \neq B_\mathsf{L}$

$\vdash_\Sigma (C_\mathsf{L}, \bot)\ \mathsf{esync}$ (by Lemma 4)

$\Gamma' \vDash_\Sigma \Theta_2, \Theta_3 :: (\Delta_1, \Delta_1'')$ (by Lemma 3-1 since $\vdash_\Sigma (C_\mathsf{L}, \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L})\ \mathsf{esync}$, $\vdash_\Sigma (C_\mathsf{L}, \bot)\ \mathsf{esync}$, and $\bot \leq \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}$)
where $\Gamma' = [b_\mathsf{S} : \bot / b_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}]\,\Gamma$

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$\Gamma' \vDash_\Sigma \Lambda_1 :: \Gamma_3$
(by Lemma 3-2 since $\vdash_\Sigma (C_\mathsf{L}, \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L})\ \mathsf{esync}$, $\vdash_\Sigma (C_\mathsf{L}, \bot)\ \mathsf{esync}$, $\bot \leq \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}$, $\mathsf{proc}(b_\mathsf{S},\,\_) \notin \Lambda_1$, and $\mathsf{unavail}(b_\mathsf{S}) \notin \Lambda_1$)

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2, \Theta_3) :: (\Delta_1, \Delta_1'')$ (by Lemma 2-2)

$[b_\mathsf{S}' : \bot / b_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}]\,\Gamma;\, \Delta_1'' \vdash_\Sigma [b_\mathsf{S}'/b_\mathsf{S}]\,P :: (b_\mathsf{L} : C_\mathsf{L})$ (by Lemma 1-3)
where b_S' *fresh*

$\Gamma';\, \Delta_1'' \vdash_\Sigma [b_\mathsf{S}/b_\mathsf{S}']\,[b_\mathsf{S}'/b_\mathsf{S}]\,P :: (b_\mathsf{L} : C_\mathsf{L})$ (by Lemma 1-4 and weakening)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: (\Delta_1, b_\mathsf{L} : C_\mathsf{L})$ (by (T-Θ_2))

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}, b_\mathsf{L}/a_\mathsf{L}]\Theta_1,\, \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: \Delta$
(by Lemma 8, since $\Gamma' \lhd \Gamma$, and Lemma 2-1 and 2-2)

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma_3$ (by Lemma 2-3 since $\mathsf{proc}(a_\mathsf{S},\,\_) \notin \Lambda_1$ and $\mathsf{unavail}(a_\mathsf{S}) \notin \Lambda_1$)

$\Gamma' \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma'$ (by (T-Λ_4))

$\Gamma' \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\; [b_\mathsf{S}/a_\mathsf{S}, b_\mathsf{L}/a_\mathsf{L}]\Theta_1,\, \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: \Gamma';\, \Delta$
(by (T-Ω) and well-formedness maintained)

Subcases: $\hat{A} \leq \hat{B}$

a_S :	$\bot$	and	b_S :	$\uparrow_\mathsf{L}^\mathsf{S} B_\mathsf{L}$
a_S :	$\bot$	and	b_S :	$\top$
a_S :	$\uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$	and	b_S :	$\top$

$\Gamma' \vDash_\Sigma \Theta_2, \Theta_3 :: (\Delta_1, \Delta_1'')$ (by Lemma 3-1 since $\vdash_\Sigma (C_\mathsf{L}, \hat{B})\ \mathsf{esync}$, $\vdash_\Sigma (C_\mathsf{L}, \hat{A})\ \mathsf{esync}$, and $\hat{A} \leq \hat{B}$)
where $\Gamma' = [b_\mathsf{S} : \hat{A} / b_\mathsf{S} : \hat{B}]\,\Gamma$

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$\Gamma' \vDash_\Sigma \Lambda_1 :: \Gamma_3$
(by Lemma 3-2 since $\vdash_\Sigma (C_\mathsf{L}, \hat{B})\ \mathsf{esync}$, $\vdash_\Sigma (C_\mathsf{L}, \hat{A})\ \mathsf{esync}$, $\hat{A} \leq \hat{B}$, $\mathsf{proc}(b_\mathsf{S},\,\_) \notin \Lambda_1$, and $\mathsf{unavail}(b_\mathsf{S}) \notin \Lambda_1$)

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2, \Theta_3) :: (\Delta_1, \Delta_1'')$ (by Lemma 2-2)

$[b_\mathsf{S}' : \hat{A} / b_\mathsf{S} : \hat{B}]\,\Gamma;\, \Delta_1'' \vdash_\Sigma [b_\mathsf{S}'/b_\mathsf{S}]\,P :: (b_\mathsf{L} : C_\mathsf{L})$ (by Lemma 1-3)
where b_S' *fresh*

$\Gamma';\, \Delta_1'' \vdash_\Sigma [b_\mathsf{S}/b_\mathsf{S}']\,[b_\mathsf{S}'/b_\mathsf{S}]\,P :: (b_\mathsf{L} : C_\mathsf{L})$ (by Lemma 1-4 and weakening)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: (\Delta_1, b_\mathsf{L} : C_\mathsf{L})$ (by (T-Θ_2))

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}, b_\mathsf{L}/a_\mathsf{L}]\Theta_1,\, \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: \Delta$ (by Lemma 8, since $\Gamma' \lhd \Gamma$, and Lemma 2-1 and 2-2)

$\Gamma' \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma_3$ (by Lemma 2-3 since $\mathsf{proc}(a_\mathsf{S},\,\_) \notin \Lambda_1$ and $\mathsf{unavail}(a_\mathsf{S}) \notin \Lambda_1$)

$\Gamma' \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma'$ (by (T-Λ_4))

$\Gamma' \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \mathsf{unavail}(b_\mathsf{S}),\, [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\; [b_\mathsf{S}/a_\mathsf{S}, b_\mathsf{L}/a_\mathsf{L}]\Theta_1,\, \mathsf{proc}(b_\mathsf{L},\,P),\, [b_\mathsf{S}/a_\mathsf{S}]\,(\Theta_2,\Theta_3) :: \Gamma';\, \Delta$
(by (T-Ω) and well-formedness maintained)

Case:

$$\begin{array}{l} \mathsf{proc}(a_\mathsf{S},\, \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}) \\ \longrightarrow \mathsf{unavail}(a_\mathsf{S}),\, a_\mathsf{S} = b_\mathsf{S} \end{array} \qquad (\text{D-ID}_\mathsf{S})$$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\ \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}),\ \Lambda_1;\ \Theta :: \Gamma;\ \Delta$ (by assumption)
for some Λ_1

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\ \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}),\ \Lambda_1;\ \Theta :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\mathsf{proc}(a_\mathsf{S},\ \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}),\ \Lambda_1;\ \Theta$ (this case)
$\longrightarrow \mathsf{unavail}(a_\mathsf{S}),\ [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\ [b_\mathsf{S}/a_\mathsf{S}]\,\Theta$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\ \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}),\ \Lambda_1 :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\ \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S}) :: \Gamma_1$ and $\Gamma \vDash_\Sigma \Lambda_1 :: \Gamma_2$ (by inversion on (T-Λ_4))
for some Γ_1 *and* Γ_2 *such that* $\Gamma = \Gamma_1, \Gamma_2$

$\Gamma_1 = a_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$ and $\Gamma \vdash_\Sigma \mathsf{fwd}\ a_\mathsf{S}\ b_\mathsf{S} :: (a_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L})$ and $\vdash_\Sigma (\uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}, \top)\ \mathsf{esync}$ (by inversion on (T-Λ_2))
for some A_L

$(b_\mathsf{S} : \hat{B}) \in \Gamma$ and $\hat{B} \leq \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$ (by inversion on (T-Id$_\mathsf{S}$))
for some $\hat{B}$

Subcase: $b_\mathsf{S} : \bot$

$\Gamma \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma_2$ (by Lemma 2-3 since $\bot \leq \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$ and $\mathsf{proc}(a_\mathsf{S},\ \_) \notin \Lambda_1$ and $\mathsf{unavail}(a_\mathsf{S}) \notin \Lambda_1$)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}) :: (a_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L})$ (by (T-Λ_3))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\ [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma$ (by (T-Λ_4))

$\Gamma \vDash_\Sigma \Theta :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Theta :: \Delta$ (by Lemma 2-2 since $\bot \leq \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\ [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\ [b_\mathsf{S}/a_\mathsf{S}]\,\Theta :: \Gamma;\ \Delta$ (by (T-Ω) and well-formedness maintained)

Subcase: $b_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$

$\Gamma \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma_2$ (by Lemma 2-3 since $\uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L} \leq \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$ and $\mathsf{proc}(a_\mathsf{S},\ \_) \notin \Lambda_1$ and $\mathsf{unavail}(a_\mathsf{S}) \notin \Lambda_1$)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}) :: (a_\mathsf{S} : \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L})$ (by (T-Λ_3))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\ [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1 :: \Gamma$ (by (T-Λ_4))

$\Gamma \vDash_\Sigma \Theta :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma [b_\mathsf{S}/a_\mathsf{S}]\,\Theta :: \Delta$ (by Lemma 2-2 since $\uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L} \leq \uparrow_\mathsf{L}^\mathsf{S} A_\mathsf{L}$)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\ [b_\mathsf{S}/a_\mathsf{S}]\,\Lambda_1;\ [b_\mathsf{S}/a_\mathsf{S}]\,\Theta :: \Gamma;\ \Delta$ (by (T-Ω) and well-formedness maintained)

Case:

$$\begin{array}{ll}\mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ !\mathsf{def}(x'_\mathsf{L} : A_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{y_\mathsf{L} : B_\mathsf{L}}, \overline{y_\mathsf{S} : B_\mathsf{S}} = P_{x'_\mathsf{L}, \overline{y_\mathsf{L}, y_\mathsf{S}}}) & \text{(D-Spawn}_{\mathsf{LL}}\text{)} \\ \quad \longrightarrow\ \mathsf{proc}(a_\mathsf{L},\ [b_\mathsf{L}/x_\mathsf{L}]Q_{x_\mathsf{L}}),\ \mathsf{proc}(b_\mathsf{L},\ [b_\mathsf{L}/x'_\mathsf{L},\ \overline{c_\mathsf{L}}/\overline{y_\mathsf{L}},\ \overline{c_\mathsf{S}}/\overline{y_\mathsf{S}}]P_{x'_\mathsf{L}, \overline{y_\mathsf{L}, y_\mathsf{S}}}),\ \mathsf{unavail}(b_\mathsf{S}) \quad (b\ \mathit{fresh}) & \end{array}$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ \Theta_2 :: \Gamma;\ \Delta$ (by assumption)
for some Θ_1 *and* Θ_2*, and where* $!\mathsf{def}(x'_\mathsf{L} : A_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{y_\mathsf{L} : B_\mathsf{L}}, \overline{y_\mathsf{S} : B_\mathsf{S}} = P_{x'_\mathsf{L}, \overline{y_\mathsf{L}, y_\mathsf{S}}})$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ \Theta_2 :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ \Theta_2$ (this case)
$\longrightarrow \mathsf{unavail}(b_\mathsf{S}),\ \Lambda;\ \Theta_1,\ \mathsf{proc}(a_\mathsf{L},\ [b_\mathsf{L}/x_\mathsf{L}]Q_{x_\mathsf{L}}),\ \mathsf{proc}(b_\mathsf{L},\ [b_\mathsf{L}/x'_\mathsf{L},\ \overline{c_\mathsf{L}}/\overline{y_\mathsf{L}},\ \overline{c_\mathsf{S}}/\overline{y_\mathsf{S}}]P_{x'_\mathsf{L}, \overline{y_\mathsf{L}, y_\mathsf{S}}}),\ \Theta_2$

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ \Theta_2 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{L},\ x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}}),\ \Theta_2 :: (\Delta_1, a_\mathsf{L} : C_\mathsf{L})$ (by Lemma 6)
for some Δ_1 *and* C_L

$\Gamma;\ \Delta'_1 \vdash_\Sigma x_\mathsf{L} \leftarrow X_\mathsf{L} \leftarrow \overline{c_\mathsf{L}}, \overline{c_\mathsf{S}}\ ;\ Q_{x_\mathsf{L}} :: (a_\mathsf{L} : C_\mathsf{L})$ (by inversion on (T-Θ_2))
for some Δ'_1

$(a_\mathsf{S} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_\mathsf{L}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))

for some $\hat{D}$

$(x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}, \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}}) \in \Sigma$
(by inversion on (T-Spawn$_{\mathsf{LL}}$) and since $\mathsf{!def}(x'_{\mathsf{L}} : A_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{y_{\mathsf{L}} : B_{\mathsf{L}}}, \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}})$)

$\Gamma;\ \Delta_2, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-Spawn$_{\mathsf{LL}}$))
where $\Delta_2 = \Delta'_1 - \overline{c_{\mathsf{L}}} : \overline{B_{\mathsf{L}}}$ *and* $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$ *for some* $\overline{\hat{B}}$ *such that* $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$

$\Gamma \vDash_\Sigma \Theta_2 :: \Delta_1, \Delta'_1$ (by inversion on (T-Θ_2))

$\vdash_\Sigma (A_{\mathsf{L}}, \top)\ \mathsf{esync}$ (by inversion on (T-Σ_2))

$\overline{y_{\mathsf{S}} : B_{\mathsf{S}}};\ \overline{y_{\mathsf{L}} : B_{\mathsf{L}}} \vdash_\Sigma P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}} :: (x'_{\mathsf{L}} : A_{\mathsf{L}})$ (by inversion on (T-Σ_2))

$\overline{y_{\mathsf{S}} : B_{\mathsf{S}}};\ \overline{c_{\mathsf{L}} : B_{\mathsf{L}}} \vdash_\Sigma [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}} :: (b_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-1 and 1-2)

$\overline{c'_{\mathsf{S}}} : \overline{\hat{B}};\ \overline{c_{\mathsf{L}} : B_{\mathsf{L}}} \vdash_\Sigma [\overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}},\ b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}} :: (b_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-3)
where $\overline{c'_{\mathsf{S}}}$ *fresh*

$\Gamma;\ \overline{c_{\mathsf{L}} : B_{\mathsf{L}}} \vdash_\Sigma [\overline{c_{\mathsf{S}}}/\overline{c'_{\mathsf{S}}},\ \overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}},\ b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}} :: (b_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-4 and weakening and since $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}}),\ \Theta_2 :: (\Delta_1, \Delta_2, b_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Θ_2) and weakening)
where $\Gamma' = \Gamma,\ b_{\mathsf{S}} : \top$

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$\Gamma';\ \Delta_2, b_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{L}}/x_{\mathsf{L}}]\,Q_{x_{\mathsf{L}}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-2 and weakening)

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}]Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}}),\ \Theta_2 :: (\Delta_1, a_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma' \vDash_\Sigma \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}]Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}}),\ \Theta_2 :: \Delta$ (by Lemma 8, since $\Gamma' \lhd \Gamma$)

$\Gamma' \vDash_\Sigma \mathsf{unavail}(b_{\mathsf{S}}) :: (b_{\mathsf{S}} : \top)$ (by (T-Λ_3))

$\Gamma' \vDash_\Sigma \mathsf{unavail}(b_{\mathsf{S}}),\ \Lambda :: \Gamma'$ (by (T-Λ_4) and weakening)

$\Gamma' \vDash_\Sigma \mathsf{unavail}(b_{\mathsf{S}}),\ \Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}]Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x'_{\mathsf{L}},\ \overline{c_{\mathsf{L}}}/\overline{y_{\mathsf{L}}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{L}}, \overline{y_{\mathsf{L}}, y_{\mathsf{S}}}}),\ \Theta_2 :: \Gamma';\ \Delta$
(by (T-Ω) and well-formedness maintained)

Case:

$$\begin{array}{l} \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \mathsf{!def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}) \\ \quad\longrightarrow\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}) \quad (b\ \textit{fresh}) \end{array} \qquad \text{(D-Spawn}_{\mathsf{LS}}\text{)}$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_2 :: \Gamma;\ \Delta$ (by assumption)
for some Θ_1 *and* Θ_2 *and where* $\mathsf{!def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y : B_i} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}})$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_2 :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_2$
$\longrightarrow \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}),\ \Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \Theta_2$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_2 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_2 :: (\Delta_1, a_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma;\ \Delta'_1 \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-Θ_2))
for some Δ'_1

$(a_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some $\hat{D}$

$(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}}) \in \Sigma$
(by inversion on (T-Spawn$_{\mathsf{LS}}$) and since $\mathsf{!def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}}, \overline{y_{\mathsf{S}}}})$)

$\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}}; \Delta'_1 \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ and $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$ for some $\overline{\hat{B}}$ such that $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$ (by inversion on (T-Spawn$_{\mathsf{LS}}$))

$\Gamma \vDash_\Sigma \Theta_2 :: \Delta_1, \Delta'_1$ (by inversion on (T-Θ_2))

$\vdash_\Sigma (A_{\mathsf{S}}, \top)\ \mathsf{esync}$ (by inversion on (T-Σ_2))

$\overline{y_{\mathsf{S}} : B_{\mathsf{S}}} \vdash_\Sigma P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} :: (x'_{\mathsf{S}} : A_{\mathsf{S}})$ (by inversion on (T-Σ_2))

$\overline{c'_{\mathsf{S}} : \hat{B}}, b_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-5 and 1-7)
where $\overline{c'_{\mathsf{S}}}$ *fresh*

$\Gamma, b_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [\overline{c_{\mathsf{S}}}/\overline{c'_{\mathsf{S}}}]\,([b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-8 and weakening and since $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$)

$\Gamma, b_{\mathsf{S}} : A_{\mathsf{S}}; \Delta'_1 \vdash_\Sigma [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-3)

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \Theta_2 :: (\Delta_1, a_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2) and weakening)
where $\Gamma' = \Gamma, b_{\mathsf{S}} : A_{\mathsf{S}}$

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$\Gamma' \vDash_\Sigma \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \Theta_2 :: \Delta$ (by Lemma 8, since $\Gamma' \lhd \Gamma$)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-Λ_2))

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}),\ \Lambda :: \Gamma'$ (by (T-Λ_4) and weakening)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}),\ \Lambda;\ \Theta_1,\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \Theta_2 :: \Gamma'; \Delta$
(by (T-Ω) and well-formedness maintained)

Case:

$$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}),\ !\mathsf{def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) \qquad \text{(D-S}{\small\text{PAWN}}_{\mathsf{SS}}\text{)}$$
$$\longrightarrow \mathsf{proc}(a_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) \quad (b\ \mathit{fresh})$$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}),\ \Lambda_1;\ \Theta :: \Gamma;\ \Delta$ (by assumption)
for some Λ_1 *and where* $!\mathsf{def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}})$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}),\ \Lambda_1;\ \Theta :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}),\ \Lambda_1;\ \Theta :: \Gamma;\ \Delta$
$\longrightarrow \mathsf{proc}(a_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}),\ \Lambda_1;\ \Theta$ (this case)

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}),\ \Lambda_1 :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}}) :: \Gamma_1$ and $\Gamma \vDash_\Sigma \Lambda_1 :: \Gamma_2$ (by inversion on (T-Λ_4))
for some Γ_1 *and* Γ_2 *such that* $\Gamma = \Gamma_1, \Gamma_2$

$\Gamma_1 = a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}}$ and $\Gamma \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{c_{\mathsf{S}}};\ Q_{x_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ and $\vdash_\Sigma (\uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}}, \top)\ \mathsf{esync}$ (by inversion on (T-Λ_2))
for some C_{L}

$(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) \in \Sigma$
(by inversion on (T-S<span>PAWN</span>$_{\mathsf{SS}}$) and since $!\mathsf{def}(x'_{\mathsf{S}} : A_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}} : B_{\mathsf{S}}} = P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}})$)

$\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma Q_{x_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ and $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$ for some $\overline{\hat{B}}$ such that $\overline{\hat{B}} \leq \overline{B_{\mathsf{S}}}$ (by inversion on (T-SPAWN$_{\mathsf{SS}}$))

$\vdash_\Sigma (A_{\mathsf{S}}, \top)\ \mathsf{esync}$ (by inversion on (T-Σ_2))

$\overline{y_{\mathsf{S}} : B_{\mathsf{S}}} \vdash_\Sigma P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} :: (x'_{\mathsf{S}} : A_{\mathsf{S}})$ (by inversion on (T-Σ_2))

$\overline{c'_{\mathsf{S}} : \hat{B}}, b_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}} :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-5 and 1-7)
where $\overline{c'_{\mathsf{S}}}$ *fresh*

$\Gamma, b_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [\overline{c_{\mathsf{S}}}/\overline{c'_{\mathsf{S}}}]\,([b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c'_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-8 and weakening and since $\overline{c_{\mathsf{S}}} : \overline{\hat{B}} \subseteq \Gamma$)

$\Gamma, b_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [b_{\mathsf{S}}/x_{\mathsf{S}}]\,Q_{x_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (by Lemma 1-7)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}) :: (b_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-Λ_2))
where $\Gamma' = \Gamma, b_{\mathsf{S}} : A_{\mathsf{S}}$

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$\Gamma' \vDash_\Sigma \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{c_{\mathsf{S}}}/\overline{y_{\mathsf{S}}}]P_{x'_{\mathsf{S}},\overline{y_{\mathsf{S}}}}),\ \Lambda_1 :: (\Gamma_2, b_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-Λ_4) and weakening)

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]Q_{x_{\mathsf{S}}}) :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} C_{\mathsf{L}})$ (by (T-Λ_2))

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\, [b_\mathsf{S}/x_\mathsf{S}]\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(b_\mathsf{S},\, [b_\mathsf{S}/x'_\mathsf{S},\, \overline{c_\mathsf{S}}/\overline{y_\mathsf{S}}] P_{x'_\mathsf{S},\overline{y_\mathsf{S}}}),\, \Lambda_1 :: \Gamma'$ (by (T-Λ_4))

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\, [b_\mathsf{S}/x_\mathsf{S}]\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(b_\mathsf{S},\, [b_\mathsf{S}/x'_\mathsf{S},\, \overline{c_\mathsf{S}}/\overline{y_\mathsf{S}}] P_{x'_\mathsf{S},\overline{y_\mathsf{S}}}),\, \Lambda_1;\ \Theta :: \Gamma'; \Delta$
(by (T-Ω) and weakening and well-formedness maintained)

Case:

$$\mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \mathsf{unavail}(a_\mathsf{S}) \qquad (\text{D-}{\downarrow}^\mathsf{S}_\mathsf{L} - \mathsf{release/detach})$$
$$\longrightarrow \mathsf{proc}(c_\mathsf{L},\, [a_\mathsf{S}/x_\mathsf{S}]\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{S},\, [a_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}})$$

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \Theta_2,\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}), \Theta_3 :: \Gamma,\, \Delta$
for some Λ_1, Θ_1, Θ_2, and Θ_3 (by assumption)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \Theta_2,\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}), \Theta_3 :: \Gamma,\, \Delta$ is well-formed
(by I.H.)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \Theta_2, \Theta_3 :: \Gamma,\, \Delta$
(by Lemma 5)

$\mathsf{unavail}(a_\mathsf{S}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \Theta_2, \Theta_3$
$\longrightarrow \mathsf{proc}(a_\mathsf{S},\, [a_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}}),\, \Lambda_1;\ \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, [a_\mathsf{S}/x_\mathsf{S}]\, Q_{x_\mathsf{S}}),\, \Theta_2, \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}),\, \Lambda_1 :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_\mathsf{S}) :: \Gamma_1$ and $\Gamma \vDash_\Sigma \Lambda_1 :: \Gamma_2$ (by inversion on (T-Λ_4))
for some Γ_1 and Γ_1 such that $\Gamma = \Gamma_1, \Gamma_2$

$\Gamma_1 = a_\mathsf{S} : \hat{B}$ (by inversion on (T-Λ_3))
for some $\hat{B}$

$\Gamma \vDash_\Sigma \Theta_1,\, \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \Theta_2, \Theta_3 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{release}\ a_\mathsf{L}\, ;\, Q_{x_\mathsf{S}}),\, \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \Theta_2, \Theta_3 :: (\Delta_1, c_\mathsf{L} : C_\mathsf{L})$ (by Lemma 6)
for some Δ_1 and C_L

$\Gamma;\ \Delta'_1 \vdash_\Sigma x_\mathsf{S} \leftarrow \mathsf{release}\, a_\mathsf{L}\, ; Q_{x_\mathsf{S}} :: (c_\mathsf{L} : C_\mathsf{L})$ and $(c_\mathsf{S} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_\mathsf{L}, \hat{D})$ esync (by inversion on (T-Θ_2))
for some Δ'_1 and $\hat{D}$

$\Delta'_1 = \Delta_2, a_\mathsf{L} : {\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S}$ and $\Gamma, x_\mathsf{S} : A_\mathsf{S};\ \Delta_2 \vdash_\Sigma Q_{x_\mathsf{S}} :: (c_\mathsf{L} : C_\mathsf{L})$ (by inversion on (T-${\downarrow}^\mathsf{S}_\mathsf{L}$L))
for some Δ_2 and A_S

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{L},\, x_\mathsf{S} \leftarrow \mathsf{detach}\ a_\mathsf{L}\, ;\, P_{x_\mathsf{S}}),\, \Theta_2, \Theta_3 :: (\Delta_1, \Delta_2, a_\mathsf{L} : {\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S})$ (by inversion on (T-Θ_2))

$\Gamma;\ \Delta''_1 \vdash_\Sigma x_\mathsf{S} \leftarrow \mathsf{detach}\, a_\mathsf{L}\, ; P_{x_\mathsf{S}} :: (a_\mathsf{L} : {\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S})$ and $\vdash_\Sigma ({\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S}, \hat{B})$ esync
for some Δ''_1 (by inversion on (T-Θ_2) and since $(a_\mathsf{S} : \hat{B}) \in \Gamma$)

$\Delta''_1 = \cdot$ and $\Gamma \vdash_\Sigma P_{x_\mathsf{S}} :: (x_\mathsf{S} : A_\mathsf{S})$ (by inversion on (T-${\downarrow}^\mathsf{S}_\mathsf{L}$R))

$\Gamma \vDash_\Sigma \Theta_2, \Theta_3 :: (\Delta_1, \Delta_2)$ (by inversion on (T-Θ_2))

Either $\hat{B} = {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}$, for some B_L, or $\hat{B} = \top$
(because $\vdash_\Sigma ({\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S}, \bot)$ esync is impossible according to the rules in Figure 16)

Subcase: *$\hat{B} = {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}$, for some B_L*
Then, $\vdash_\Sigma ({\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S}, {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L})$ esync. *Consequently, the type A_S of the continuation P_{x_S} must be ${\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}$.*

$A_\mathsf{S} = {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}$ (by $\vdash_\Sigma ({\downarrow}^\mathsf{S}_\mathsf{L} A_\mathsf{S}, \hat{B})$ esync)

$\vdash_\Sigma ({\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}, \top)$ esync (by inversion on (T-$\textsc{Esync}_{{\downarrow}^\mathsf{S}_\mathsf{L}}$-1) since $\vdash_\Sigma ({\downarrow}^\mathsf{S}_\mathsf{L}{\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}, {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L})$ esync)

$\Gamma, a'_\mathsf{S} : {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L} \vdash_\Sigma [a'_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}} :: (a'_\mathsf{S} : {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L})$ (by Lemma 1-5)
where a'_S fresh

$\Gamma \vdash_\Sigma [a_\mathsf{S}/a'_\mathsf{S}]\, ([a'_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}}) :: (a_\mathsf{S} : {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L})$ (by Lemma 1-6 since $(a_\mathsf{S} : {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L}) \in \Gamma$)

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\, [a_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}}) :: (a_\mathsf{S} : {\uparrow}^\mathsf{S}_\mathsf{L} B_\mathsf{L})$ (by (T-Λ_2))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_\mathsf{S},\, [a_\mathsf{S}/x_\mathsf{S}]\, P_{x_\mathsf{S}}),\, \Lambda_1 :: \Gamma$ (by (T-Λ_4) and since $\mathsf{proc}(a_\mathsf{S},\, \_) \notin \Lambda_1$ by well-formedness of Λ)

$\Gamma, a''_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}};\ \Delta_2 \vdash_\Sigma [a''_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-7)
where a''_{S} *fresh*

$\Gamma;\ \Delta_2 \vdash_\Sigma [a_{\mathsf{S}}/a''_{\mathsf{S}}]\,([a''_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}) :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-8 since $(a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} B_{\mathsf{L}}) \in \Gamma$)

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: \Gamma;\ \Delta$
(by (T-Ω) and well-formedness maintained)

Subcase: $\hat{B} = \top$
Then, $\vdash_\Sigma (\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}, \top)$ esync. *Consequently,* Γ *must be updated as follows:* $[a_{\mathsf{S}} : A_{\mathsf{S}}/a_{\mathsf{S}} : \top]\,\Gamma$.

$\vdash_\Sigma (\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}, A_{\mathsf{S}})\ \mathsf{esync}$ (by coinduction on rules in Figure 16, rule (T-ESYNC$_{\downarrow^{\mathsf{S}}_{\mathsf{L}}}$-1) in particular)

$\Gamma' \vDash_\Sigma \Theta_2, \Theta_3 :: (\Delta_1, \Delta_2)$ and $\Gamma' \vDash_\Sigma \Lambda_1 :: \Gamma_2$
where $\Gamma' = [a_{\mathsf{S}} : A_{\mathsf{S}}/a_{\mathsf{S}} : \top]\,\Gamma$ (by Lemma 3-1 and 3-2 since $\vdash_\Sigma (\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}, \top)$ esync and $\vdash_\Sigma (\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}, A_{\mathsf{S}})$ esync)
(and $A_{\mathsf{S}} \leq \top$ and $\mathsf{proc}(a_{\mathsf{S}},\ \_) \notin \Lambda_1$ and $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda_1$ by well-formedness of Λ)

$\Gamma' \lhd \Gamma$ (by Definition 2 since $\lhd$ is reflexive)

$[a'_{\mathsf{S}} : A_{\mathsf{S}}/a_{\mathsf{S}} : \top]\,\Gamma \vdash_\Sigma [a'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-7)
where a'_{S} *fresh*

$\Gamma' \vdash_\Sigma [a_{\mathsf{S}}/a'_{\mathsf{S}}]\,([a'_{\mathsf{S}}/a_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}) :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-8 and weakening)

$\Gamma', a''_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [a''_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}} :: (a''_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-5)
where a''_{S} *fresh*

$\Gamma' \vdash_\Sigma [a_{\mathsf{S}}/a''_{\mathsf{S}}]\,([a''_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}) :: (a_{\mathsf{S}} : A_{\mathsf{S}})$ (by Lemma 1-6 since $(a_{\mathsf{S}} : A_{\mathsf{S}}) \in \Gamma'$)

$\vdash_\Sigma (A_{\mathsf{S}}, \top)\ \mathsf{esync}$ (by inversion on (T-ESYNC$_{\downarrow^{\mathsf{S}}_{\mathsf{L}}}$-2) since $\vdash_\Sigma (\downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}}, \top)$ esync)

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}) :: (a_{\mathsf{S}} : A_{\mathsf{S}})$ (by (T-Λ_2))

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}),\ \Lambda_1 :: \Gamma'$ (by (T-Λ_4) and since $\mathsf{proc}(a_{\mathsf{S}},\ \_) \notin \Lambda_1$ by well-formedness of Λ)

$[a'''_{\mathsf{S}} : A_{\mathsf{S}}/a_{\mathsf{S}} : \top]\,\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [a'''_{\mathsf{S}}/a_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-7)
where a'''_{S} *fresh*

$\Gamma', x_{\mathsf{S}} : A_{\mathsf{S}} \vdash_\Sigma [a_{\mathsf{S}}/a'''_{\mathsf{S}}]\,([a'''_{\mathsf{S}}/a_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}) :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-8 and weakening)

$\Gamma';\ \Delta_2 \vdash_\Sigma [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-8)

$\Gamma' \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2) and weakening)

$\Gamma' \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: \Delta$ (by Lemma 8, since $\Gamma' \lhd \Gamma$)

$\Gamma' \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, P_{x_{\mathsf{S}}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{S}}/x_{\mathsf{S}}]\, Q_{x_{\mathsf{S}}}),\ \Theta_2, \Theta_3 :: \Gamma';\ \Delta$
(by (T-Ω) and well-formedness maintained)

Case:

$$\mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}) \qquad (\text{D-}\uparrow^{\mathsf{S}}_{\mathsf{L}} - \mathsf{acquire/accept})$$
$$\longrightarrow \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \mathsf{unavail}(a_{\mathsf{S}})$$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \Theta_2 :: \Gamma;\ \Delta$ (by assumption)
for some Λ_1*,* Θ_1*, and* Θ_2

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \Theta_2 :: \Gamma;\ \Delta$ is well-formed (by I.H.)

$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \Theta_2$ (this case)
$\longrightarrow \mathsf{unavail}(a_{\mathsf{S}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \Theta_2$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}),\ \Lambda_1 :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\ ;\ P_{x_{\mathsf{L}}}) :: \Gamma_1$ and $\Gamma \vDash_\Sigma \Lambda_1 :: \Gamma_2$ (by inversion on (T-Λ_4))
for some Γ_1 *and* Γ_2 *such that* $\Gamma = \Gamma_1, \Gamma_2$

$\Gamma_1 = a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$ and $\Gamma \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{accept}\, a_{\mathsf{S}};\ P_{x_{\mathsf{L}}} :: (a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})$ and $\vdash_\Sigma (\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}, \top)\ \mathsf{esync}$ (by inversion on (T-Λ_2))

for some A_{L}

$\Gamma; \cdot \vdash_\Sigma P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ (by inversion on (T-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$R))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}) :: (a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})$ (by (T-Λ_3))

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\ \Lambda_1 :: \Gamma$ (by (T-Λ_4) and since $\mathsf{unavail}(a_{\mathsf{S}}) \notin \Lambda_1$ by well-formedness of Λ)

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \Theta_2 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ a_{\mathsf{S}}\ ;\ Q_{x_{\mathsf{L}}}),\ \Theta_2 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma;\ \Delta_1' \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, a_{\mathsf{S}}\, ; Q_{x_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' *and* $\hat{D}$

$\Gamma \vDash_\Sigma \Theta_2 :: (\Delta_1, \Delta_1')$ (by inversion on (T-Θ_2))

$\Gamma;\ \Delta_1', x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{x_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$L) and since $(a_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}) \in \Gamma$)

$\vdash_\Sigma (A_{\mathsf{L}}, \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}})\ \mathsf{esync}$ (by inversion on (T-ESYNC$_{\uparrow^{\mathsf{S}}_{\mathsf{L}}}$) since $\vdash_\Sigma (\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}, \top)\ \mathsf{esync}$)

$\Gamma; \cdot \vdash_\Sigma [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ (by Lemma 1-1 and Corollary 1)

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \Theta_2 :: (\Delta_1, \Delta_1', a_{\mathsf{L}} : A_{\mathsf{L}})$ (by (T-Θ_2) and Corollary 1)

$\Gamma;\ \Delta_1', a_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-2 and Corollary 1)

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \Theta_2 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \Theta_2 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \mathsf{unavail}(a_{\mathsf{S}}),\ \Lambda_1;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, Q_{x_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ [a_{\mathsf{L}}/x_{\mathsf{L}}]\, P_{x_{\mathsf{L}}}),\ \Theta_2 :: \Gamma;\ \Delta$
(by (T-Ω) and well-formedness is maintained)

Case:

$$\begin{array}{l} \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}) \\ \quad \longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ Q) \end{array} \qquad (\text{D-}\mathbf{1})$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_2,\ \Theta_3$
$\quad \longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_2,\ \Theta_3 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{wait}\ a_{\mathsf{L}}\ ;\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma \mathsf{wait}\, a_{\mathsf{L}}\, ; Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' *and* $\hat{D}$

$\Gamma; \Delta_2 \vdash_\Sigma Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\mathbf{1}_{\mathrm{L}}$))
for some Δ_2 *such that* $\Delta_1' = \Delta_2, a_{\mathsf{L}} : \mathbf{1}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : \mathbf{1})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma \mathsf{close}\, a_{\mathsf{L}} :: (a_{\mathsf{L}} : \mathbf{1})$ and $(a_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\vdash_\Sigma (\mathbf{1}, \hat{B})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_3 *and* $\hat{B}$

$\Delta_3 = (\cdot)$ (by inversion on (T-$\mathbf{1}_{\mathrm{R}}$))

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2)$ (by inversion on (T-Θ_2))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P) \qquad \text{(D-}\otimes\text{)}$$
$$\longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P)$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_2,\ \Theta_3$
$\longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, a_{\mathsf{L}}\, ; Q_{y_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' *and* $\hat{D}$

$\Gamma; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}}, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\otimes_{\mathsf{L}}$))
for some Δ_2, A_{L}, *and* B_{L} *such that* $\Delta_1' = \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma \mathsf{send}\, a_{\mathsf{L}}\, b_{\mathsf{L}}\, ; P :: (a_{\mathsf{L}} : A_{\mathsf{L}} \otimes B_{\mathsf{L}})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (A_{\mathsf{L}} \otimes B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_3 *and* $\hat{A}$

$\Gamma; \Delta_4 \vdash_\Sigma P :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by inversion on (T-$\otimes_{\mathsf{R}}$))
for some Δ_4 *such that* $\Delta_3 = \Delta_4, b_{\mathsf{L}} : A_{\mathsf{L}}$

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_4, b_{\mathsf{L}} : A_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\vdash_\Sigma (B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-$\text{E}\textsc{sync}_\otimes$))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, b_{\mathsf{L}} : A_{\mathsf{L}}, a_{\mathsf{L}} : B_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}}, b_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{L}}/y_{\mathsf{L}}]\, Q_{y_{\mathsf{L}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-2)

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P) \qquad \text{(D-}\exists\text{)}$$
$$\longrightarrow\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P)$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_2,\ \Theta_3$
$\longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;\ Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6) *for some* Δ_1 *and* C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\, a_{\mathsf{L}}\, ; Q_{y_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))

for some Δ_1' *and* $\hat{D}$

$\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}}; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q_{y_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\exists_{\mathrm{L}}$))

for some Δ_2, A_{S}, *and* B_{L} *such that* $\Delta_1' = \Delta_2, a_{\mathsf{L}} : \exists A_{\mathsf{S}}.B_{\mathsf{L}}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ; P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : \exists A_{\mathsf{S}}.B_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma \mathsf{send}\, a_{\mathsf{L}}\, b_{\mathsf{S}}\, ; P :: (a_{\mathsf{L}} : \exists A_{\mathsf{S}}.B_{\mathsf{L}})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (\exists A_{\mathsf{S}}.B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))

for some Δ_3 *and* $\hat{A}$

$\Gamma; \Delta_3 \vdash_\Sigma P :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ and $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\hat{B} \leq A_{\mathsf{S}}$ (by inversion on (T-$\exists_{\mathrm{R}}$))

for some $\hat{B}$

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_3)$ (by inversion on (T-Θ_2))

$\vdash_\Sigma (B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-$\mathrm{E}\textsc{sync}_\exists$))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma, b_{\mathsf{S}}' : \hat{B}; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{S}}'/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-3)

where b_{S}' *fresh*

$\Gamma; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{S}}/b_{\mathsf{S}}']\, ([b_{\mathsf{S}}'/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}}) :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 1-4 since $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$)

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\, Q_{y_{\mathsf{S}}}),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\begin{aligned}&\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}) \\ &\longrightarrow \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}})\end{aligned} \qquad (\text{D-}\multimap)$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)

for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3$

$\longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)

for some Δ_1 *and* C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma \mathsf{send}\, a_{\mathsf{L}}\, b_{\mathsf{L}}\, ; Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))

for some Δ_1' *and* $\hat{D}$

$\Gamma; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\multimap_{\mathrm{L}}$))

for some Δ_2, A_{L}, *and* B_{L} *such that* $\Delta_1' = \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, b_{\mathsf{L}} : A_{\mathsf{L}}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ; P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}}, b_{\mathsf{L}} : A_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, a_{\mathsf{L}}\, ; P_{y_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}} \multimap B_{\mathsf{L}})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (A_{\mathsf{L}} \multimap B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))

for some Δ_3 *and* $\hat{A}$

$\Gamma; \Delta_3, y_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma P_{y_{\mathsf{L}}} :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by inversion on (T-$\multimap_{\mathrm{R}}$))

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_3, b_{\mathsf{L}} : A_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3, b_{\mathsf{L}} : A_{\mathsf{L}} \vdash_\Sigma [b_{\mathsf{L}}/y_{\mathsf{L}}]\, P_{y_{\mathsf{L}}} :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by Lemma 1-2)

$\vdash_\Sigma (B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-$\mathrm{E}\textsc{sync}_\multimap$))

$\Gamma \models_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]P_{y_{\mathsf{L}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\begin{aligned}&\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}) \\ &\longrightarrow \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}})\end{aligned} \qquad \text{(D-}\Pi\text{)}$$

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3$
$\longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \models_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \models_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by inversion on (T-Ω))

$\Gamma \models_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q),\ \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 *and* C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ;Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' *and* $\hat{D}$

$\Gamma; \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$ and $\hat{B} \leq A_{\mathsf{S}}$ (by inversion on (T-Π_{L}))
for some Δ_2, $\hat{B}$, A_{S}, *and* B_{L} *such that* $\Delta_1' = \Delta_2, a_{\mathsf{L}} : \Pi A_{\mathsf{S}}.B_{\mathsf{L}}$

$\Gamma \models_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : \Pi A_{\mathsf{S}}.B_{\mathsf{L}})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\ ;P_{y_{\mathsf{S}}} :: (a_{\mathsf{L}} : \Pi A_{\mathsf{S}}.B_{\mathsf{L}})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (\Pi A_{\mathsf{S}}.B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_3 *and* $\hat{A}$

$\Gamma, y_{\mathsf{S}} : A_{\mathsf{S}}; \Delta_3 \vdash_\Sigma P_{y_{\mathsf{S}}} :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by inversion on (T-Π_{R}))

$\Gamma \models_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_3)$ (by inversion on (T-Θ_2))

$\Gamma, b_{\mathsf{S}}' : \hat{B}; \Delta_3 \vdash_\Sigma [b_{\mathsf{S}}'/y_{\mathsf{S}}]\, P_{y_{\mathsf{S}}} :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by Lemma 1-3)
where b_{S}' *fresh*

$\Gamma; \Delta_3 \vdash_\Sigma [b_{\mathsf{S}}/b_{\mathsf{S}}']\,([b_{\mathsf{S}}'/y_{\mathsf{S}}]\, P_{y_{\mathsf{S}}}) :: (a_{\mathsf{L}} : B_{\mathsf{L}})$ (by Lemma 1-4 since $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$)

$\vdash_\Sigma (B_{\mathsf{L}}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-$\textsc{Esync}_\Pi$))

$\Gamma \models_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : B_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]P_{y_{\mathsf{S}}}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\begin{aligned}&\mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;P) \\ &\longrightarrow \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P)\end{aligned} \qquad \text{(D-}\oplus\text{)}$$

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;P),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, *and* Θ_3

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ;P),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; P),\ \Theta_2,\ \Theta_3$
$\quad \longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\ \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 and C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q} :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' and $\hat{D}$

$(\forall i)\ \ \Gamma;\ \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}_i} \vdash_\Sigma Q_i :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\oplus_{\mathrm{L}}$))
for some Δ_2 and $\overline{A_{\mathsf{L}}}$ such that $\Delta_1' = \Delta_2, a_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma a_{\mathsf{L}}.l_h\ ; P :: (a_{\mathsf{L}} : \oplus\{\overline{l : A_{\mathsf{L}}}\})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (\oplus\{\overline{l : A_{\mathsf{L}}}\}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_3 and $\hat{A}$

$\Gamma; \Delta_3 \vdash_\Sigma P :: (a_{\mathsf{L}} : A_{\mathsf{L}\,h})$ (by inversion on (T-$\oplus_{\mathrm{R}}$))

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_3)$ (by inversion on (T-Θ_2))

$(\forall i)\ \vdash_\Sigma (A_{\mathsf{L}_i}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-$\mathrm{E}\textsc{sync}_\oplus$))

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}\,h})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q_h),\ \mathsf{proc}(a_{\mathsf{L}},\ P),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

Case:

$$\begin{aligned}&\mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P})\\ &\longrightarrow \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h)\end{aligned} \qquad (\text{D-}\&)$$

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_3 :: \Gamma; \Delta$ (by assumption)
for some Θ_1, Θ_2, and Θ_3

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \Theta_2,\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_3 :: \Gamma; \Delta$ is well-formed (by I.H.)

$\Gamma \vDash_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by Lemma 5)

$\Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_2,\ \Theta_3$
$\quad \longrightarrow \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h),\ \Theta_2,\ \Theta_3$ (this case)

$\Gamma \vDash_\Sigma \Lambda :: \Gamma$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by inversion on (T-Ω))

$\Gamma \vDash_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ a_{\mathsf{L}}.l_h\ ; Q),\ \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by Lemma 6)
for some Δ_1 and C_{L}

$\Gamma; \Delta_1' \vdash_\Sigma a_{\mathsf{L}}.l_h\ ; Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ and $(c_{\mathsf{S}} : \hat{D}) \in \Gamma$ and $\vdash_\Sigma (C_{\mathsf{L}}, \hat{D})\ \mathsf{esync}$ (by inversion on (T-Θ_2))
for some Δ_1' and $\hat{D}$

$\Gamma;\ \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}\,h} \vdash_\Sigma Q :: (c_{\mathsf{L}} : C_{\mathsf{L}})$ (by inversion on (T-$\&_{\mathrm{L}}$))
for some Δ_2 and $\overline{A_{\mathsf{L}}}$ such that $\Delta_1' = \Delta_2, a_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\}$

$\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})$ (by inversion on (T-Θ_2))

$\Gamma; \Delta_3 \vdash_\Sigma \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P} :: (a_{\mathsf{L}} : \&\{\overline{l : A_{\mathsf{L}}}\})$ and $(a_{\mathsf{S}} : \hat{A}) \in \Gamma$ and $\vdash_\Sigma (\&\{\overline{l : A_{\mathsf{L}}}\}, \hat{A})\ \mathsf{esync}$
for some Δ_3 and $\hat{A}$ (by inversion on (T-Θ_2))

$(\forall i)\Gamma; \Delta_3 \vdash_\Sigma P_i :: (a_{\mathsf{L}} : A_{\mathsf{L}_i})$ (by inversion on (T-$\&_{\mathrm{R}}$))

$\Gamma \vDash_\Sigma \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, \Delta_3)$ (by inversion on (T-Θ_2))

$(\forall i)\ \vdash_\Sigma (A_{\mathsf{L}_i}, \hat{A})\ \mathsf{esync}$ (by inversion on (T-ESYNC$_\&$))

$\Gamma \models_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P_h),\ \Theta_2,\ \Theta_3 :: (\Delta_1, \Delta_2, a_{\mathsf{L}} : A_{\mathsf{L}\,h})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h),\ \Theta_2,\ \Theta_3 :: (\Delta_1, c_{\mathsf{L}} : C_{\mathsf{L}})$ (by (T-Θ_2))

$\Gamma \models_\Sigma \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h),\ \Theta_2,\ \Theta_3 :: \Delta$ (by Lemma 8)

$\Gamma \models_\Sigma \Lambda;\ \Theta_1,\ \mathsf{proc}(c_{\mathsf{L}},\ Q),\ \mathsf{proc}(a_{\mathsf{L}},\ P_h),\ \Theta_2,\ \Theta_3 :: \Gamma; \Delta$ (by (T-Ω) and well-formedness is maintained)

□

D.3 Progress

The progress theorem relies on the notions of a *poised* and *blocked* process (see Definition 3 and Definition 4) and expresses that being blocked is the *only* way the whole configuration may be stuck [27]. Case (2-c) captures the scenario where a blocked process cannot proceed because the shared channel is unavailable. A successful acquire, on the other hand, is represented as part of case (2-a).

Theorem 4 (Progress). *If* $\Gamma \models_\Sigma \Lambda; \Theta :: \Gamma; \Delta$*, then either*

1. $\Lambda \longrightarrow \Lambda'$*, for some* Λ'*, or*
2. Λ *is poised and*
  (a) $\Lambda;\ \Theta \longrightarrow \Lambda';\ \Theta'$*, for some* Λ' *and* Θ'*, or*
  (b) Θ *is poised, or*
  (c) some process in Θ *is blocked along* a_{S} *and* $\mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$.

Proof.

$\Gamma \models_\Sigma \Lambda; \Theta :: \Gamma; \Delta$ (by assumption)

$\Gamma \models_\Sigma \Lambda :: \Gamma$ and $\Gamma \models_\Sigma \Theta :: \Delta$ (by inversion on (T-Ω))

We first show that either $\Lambda \longrightarrow \Lambda'$, for some Λ', or that Λ is poised. We proceed by induction on $\Gamma \models_\Sigma \Lambda :: \Gamma_1$, where $\Gamma_1 \subseteq \Gamma$:

Case: $\Gamma \models_\Sigma (\cdot) :: (\cdot)$

$(\cdot)$ is poised (by Definition 3)

Case: $\Gamma \models_\Sigma \mathsf{proc}(a_{\mathsf{S}},\ P_{a_{\mathsf{S}}}) :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$, for some a_{S}, $P_{a_{\mathsf{S}}}$, and A_{L}

We proceed by case analysis on $\Gamma \vdash_\Sigma P_{a_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$:

Subcase: $\Gamma \vdash_\Sigma \mathsf{fwd}\ a_{\mathsf{S}}\, b_{\mathsf{S}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$, for some $(b_{\mathsf{S}} : \hat{A}) \in \Gamma$ such that $\hat{A} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$

$\mathsf{proc}(a_{\mathsf{S}},\ \mathsf{fwd}\ a_{\mathsf{S}}\ b_{\mathsf{S}}),\ \Lambda_1 \longrightarrow \mathsf{unavail}(a_{\mathsf{S}}),\ [b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Lambda_1$ (by D-ID$_{\mathsf{S}}$)

Subcase: $\Gamma \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{y_{\mathsf{S}}}\,;\ Q_{x_{\mathsf{S}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$

$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{b}\,;\ Q_{x_{\mathsf{S}}}),\ \Lambda_1 \longrightarrow \mathsf{proc}(a_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]\,Q_{x_{\mathsf{S}}}),\ \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x'_{\mathsf{S}},\ \overline{b}/\overline{y}]P_{x'_{\mathsf{S}},\overline{y}}),\ \Lambda_1$
where b *fresh* (by D-SPAWN$_{\mathsf{SS}}$)

Subcase: $\Gamma \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{accept}\, a_{\mathsf{S}}\,;P_{x_{\mathsf{L}}} :: (a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})$

$\mathsf{proc}(a_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ a_{\mathsf{S}}\,;P_{x_{\mathsf{L}}}),\ \Lambda_1$ is poised (by Definition 3)

Case: $\Gamma \models_\Sigma \mathsf{unavail}(a_{\mathsf{S}}) :: (a_{\mathsf{S}} : \hat{A})$, for some a_{S} and $\hat{A}$

$\mathsf{unavail}(a_{\mathsf{S}})$ is poised (by Definition 3)

Case: $\Gamma \vDash_\Sigma \Lambda_1, \Lambda_2 :: \Gamma_1, \Gamma_2$, for some Λ_1, Λ_2, Γ_1, and Γ_2, such that $\Gamma = \Gamma_1, \Gamma_2$

Either $\Lambda_1 \longrightarrow \Lambda_1'$, for some Λ_1', or Λ_1 is poised, or $\Lambda_2 \longrightarrow \Lambda_2'$, for some Λ_2', or Λ_2 is poised. (by I.H.)

Subcase: $\Lambda_1 \longrightarrow \Lambda_1'$, for some Λ_1' and Λ_2 is poised

$\Lambda_1, \Lambda_2 \longrightarrow \Lambda_1', \Lambda_2$

Subcase: $\Lambda_1 \longrightarrow \Lambda_1'$, for some Λ_1' and $\Lambda_2 \longrightarrow \Lambda_2'$, for some Λ_2'

$\Lambda_1, \Lambda_2 \longrightarrow \Lambda_1', \Lambda_2'$

Subcase: Λ_1 is poised and $\Lambda_2 \longrightarrow \Lambda_2'$, for some Λ_2'

$\Lambda_1, \Lambda_2 \longrightarrow \Lambda_1, \Lambda_2'$

Subcase: Λ_1 is poised and Λ_2 is poised

Λ_1, Λ_2 is poised

Having proved that either $\Lambda \longrightarrow \Lambda'$, for some Λ', or that Λ is poised, we assume that Λ is poised and proceed by induction on $\Gamma \vDash_\Sigma \Theta :: \Delta$:

Case: $\Gamma \vDash_\Sigma (\cdot) :: (\cdot)$

$(\cdot)$ is poised (by Definition 3)

Case: $\Gamma \vDash_\Sigma \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta_1 :: (\Delta_1, a_{\mathsf{L}} : A_{\mathsf{L}})$, for some a_{L}, $P_{a_{\mathsf{S}}}$, Θ_1, Δ_1, and A_{L}

$\Gamma; \Delta_1' \vdash_\Sigma P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$ and $\Gamma \vDash_\Sigma \Theta_1 :: \Delta_1, \Delta_1'$ (by inversion on (T-Θ_2))
for some Δ_1'

Either $\Lambda; \Theta_1 \longrightarrow \Lambda'; \Theta_1'$, for some Λ' and Θ', or Θ_1 is poised or some process in Θ_1 is blocked along a_{S} and $\mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$. (by I.H.)

Subcase: $\Lambda; \Theta_1 \longrightarrow \Lambda'; \Theta_1'$

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta_1 \longrightarrow \Lambda'; \mathsf{proc}(a_{\mathsf{L}},\ P_{a_{\mathsf{L}}}),\ \Theta_1'$

Subcase: Θ_1 is poised

We proceed by case analysis on $\Gamma; \Delta_1' \vdash_\Sigma P_{a_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$:

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma \mathsf{fwd}\ a_{\mathsf{L}}\, b_{\mathsf{L}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$, for some $b_{\mathsf{L}} : A_{\mathsf{L}}$ such that $\Delta_1' = a_{\mathsf{L}} : A_{\mathsf{L}}$

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}}),\ \Theta_1 \longrightarrow [b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Lambda; [b_{\mathsf{S}}/a_{\mathsf{S}}]\,\Theta_1$ (by D-Id$_{\mathsf{L}}$)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{b}\,;\ Q_{x_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow X_{\mathsf{L}} \leftarrow \overline{b}\,;\ Q_{x_{\mathsf{L}}}),\ \Theta_1 \longrightarrow \mathsf{unavail}(b_{\mathsf{S}}), \Lambda; \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}]\,Q_{x_{\mathsf{L}}}), \mathsf{proc}(b_{\mathsf{L}},\ [b_{\mathsf{L}}/x_{\mathsf{L}}',\ \overline{b}/\overline{y}]P_{x_{\mathsf{L}}', \overline{y}}), \Theta_1$
where b fresh (by D-Spawn$_{\mathsf{LL}}$)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{b}\,;\ Q_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow X_{\mathsf{S}} \leftarrow \overline{b}\,;\ Q_{x_{\mathsf{S}}}),\ \Theta_1 \longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}]\,Q_{x_{\mathsf{S}}}), \mathsf{proc}(b_{\mathsf{S}},\ [b_{\mathsf{S}}/x_{\mathsf{S}}',\ \overline{b}/\overline{y}]P_{x_{\mathsf{S}}', \overline{y}}), \Theta_1$
where b fresh (by D-Spawn$_{\mathsf{LS}}$)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma x_{\mathsf{L}} \leftarrow \mathsf{acquire}\, c_{\mathsf{S}}\,; Q_{x_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$, for some $(c_{\mathsf{S}} : \hat{C}) \in \Gamma$ such that $\hat{C} \leq \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$, for some B_{L}

Either there exists a $\mathsf{proc}(c_{\mathsf{S}},\ P_{c_{\mathsf{S}}})$ in Λ or an $\mathsf{unavail}(c_{\mathsf{S}})$ in Λ. (by Lemma 10-1)

Subsubsubcase: There exists a Λ_1 such that $\Lambda = \mathsf{proc}(c_{\mathsf{S}},\ P_{c_{\mathsf{S}}}), \Lambda_1$:

$\mathsf{proc}(c_{\mathsf{S}},\ P_{c_{\mathsf{S}}})$ is poised (by Definition 3 since Λ is poised)

$\mathsf{proc}(c_{\mathsf{S}},\ P_{c_{\mathsf{S}}}) = \mathsf{proc}(c_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{S}}\ ;P'_{x_{\mathsf{L}}})$ (by Definition 3)

$\mathsf{proc}(c_{\mathsf{S}},\ x_{\mathsf{L}} \leftarrow \mathsf{accept}\ c_{\mathsf{S}}\ ;P'_{x_{\mathsf{L}}}), \Lambda_1;\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ c_{\mathsf{S}}\ ;Q_{x_{\mathsf{L}}}), \Theta_1$
$\longrightarrow \mathsf{unavail}(c_{\mathsf{S}}), \Lambda_1;\ \mathsf{proc}(a_{\mathsf{L}},\ [c_{\mathsf{L}}/x_{\mathsf{L}}]\ Q_{x_{\mathsf{L}}}), \mathsf{proc}(c_{\mathsf{L}},\ [c_{\mathsf{L}}/x_{\mathsf{L}}]\ P'_{x_{\mathsf{L}}}), \Theta_1$ (by D-$\uparrow^{\mathsf{S}}_{\mathsf{L}}$ – $\mathsf{acquire/accept}$)

Subsubsubcase: There exists a Λ_1 such that $\Lambda = \mathsf{unavail}(c_{\mathsf{S}}), \Lambda_1$:

$\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ c_{\mathsf{S}})$ in Θ is blocked along c_{S} and $\mathsf{unavail}(c_{\mathsf{S}}) \in \Lambda$ (by Definition 4)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : {\downarrow^{\mathsf{S}}_{\mathsf{L}}} C_{\mathsf{S}} \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{release}\ c_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$, for some C_{S} and Δ_2 such that $\Delta'_1 = \Delta_2, c_{\mathsf{L}} : {\downarrow^{\mathsf{S}}_{\mathsf{L}}} C_{\mathsf{S}}$

There exist Θ_2, Θ'_2, and $\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta'_2$ and (by Lemma 10-2)
there exists a Λ_1 and $\mathsf{unavail}(c_{\mathsf{S}})$ such that $\Lambda = \mathsf{unavail}(c_{\mathsf{S}}), \Lambda_1$. (by well-formedness of configuration)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ c_{\mathsf{L}}\ ; P'_{x_{\mathsf{S}}})$ (by Definition 3 and inversion on process typing)

$\mathsf{unavail}(c_{\mathsf{S}}), \Lambda_1;\ \mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{release}\ c_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}}), \Theta_2,\ \mathsf{proc}(c_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ c_{\mathsf{L}}\ ; P'_{x_{\mathsf{S}}}), \Theta'_2$
$\longrightarrow \mathsf{proc}(c_{\mathsf{S}},\ [c_{\mathsf{S}}/x_{\mathsf{S}}]\ P'_{x_{\mathsf{S}}}), \Lambda_1;\ \mathsf{proc}(a_{\mathsf{L}},\ [c_{\mathsf{S}}/x_{\mathsf{S}}]\ Q_{x_{\mathsf{S}}}), \Theta_2, \Theta'_2$ (by D-$\downarrow^{\mathsf{S}}_{\mathsf{L}}$ – $\mathsf{release/detach}$)

Subsubcase: $\Gamma; \cdot \vdash_\Sigma x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$

$\mathsf{proc}(a_{\mathsf{L}},\ x_{\mathsf{S}} \leftarrow \mathsf{detach}\ a_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}}), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : \mathbf{1} \vdash_\Sigma \mathsf{wait}\ c_{\mathsf{L}}\ ; Q :: (a_{\mathsf{L}} : A_{\mathsf{L}})$, for some Δ_2 such that $\Delta'_1 = \Delta_2, c_{\mathsf{L}} : \mathbf{1}$

There exist Θ_2, Θ'_2, and $\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta'_2$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{close}\ c_{\mathsf{L}}\ ; P'_{x_{\mathsf{S}}})$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ \mathsf{wait}\ c_{\mathsf{L}}\ ; Q),\ \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{close}\ c_{\mathsf{L}}), \Theta'_2$
$\longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\ Q),\ \Theta_2, \Theta'_2$ (by D-$\mathbf{1}$)

Subsubcase: $\Gamma; \cdot \vdash_\Sigma \mathsf{close}\ a_{\mathsf{L}} :: (a_{\mathsf{L}} : \mathbf{1})$

$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{close}\ a_{\mathsf{L}}), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : B_{\mathsf{L}} \otimes C_{\mathsf{L}} \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; Q_{y_{\mathsf{L}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{L}, C_{L}, and Δ_2 such that $\Delta'_1 = \Delta_2, c_{\mathsf{L}} : B_{\mathsf{L}} \otimes C_{\mathsf{L}}$

There exist Θ_2, Θ'_2, and $\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta'_2$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{L}}\ ; P')$,
for some $b_{\mathsf{L}} : B_{\mathsf{L}}$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{L}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; Q_{y_{\mathsf{L}}}),\ \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{L}}\ ; P'), \Theta'_2$
$\longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{L}}/y_{\mathsf{L}}]\ Q_{y_{\mathsf{L}}}),\ \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P'), \Theta'_2$ (by D-$\otimes/\exists$)

Subsubcase: $\Gamma; \Delta_2, b_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; P :: (a_{\mathsf{L}} : B_{\mathsf{L}} \otimes C_{\mathsf{L}})$,
for some B_{L}, C_{L}, and Δ_2 such that $A_{\mathsf{L}} = B_{\mathsf{L}} \otimes C_{\mathsf{L}}$ and $\Delta'_1 = \Delta_2, b_{\mathsf{L}} : B_{\mathsf{L}}$

$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{L}}\ ; P), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : (\exists B_{\mathsf{S}}.C_{\mathsf{L}}) \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; Q_{y_{\mathsf{S}}} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{S}, C_{L}, and Δ_2 such that $\Delta'_1 = \Delta_2, c_{\mathsf{L}} : \exists B_{\mathsf{S}}.C_{\mathsf{L}}$

There exist Θ_2, Θ'_2, and $\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}), \Theta'_2$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\ P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{S}}\ ; P')$,
for some $b_{\mathsf{S}} : B_{\mathsf{S}}$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\ y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\ ; Q_{y_{\mathsf{S}}}),\ \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{S}}\ ; P'), \Theta'_2$
$\longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\ [b_{\mathsf{S}}/y_{\mathsf{S}}]\ Q_{y_{\mathsf{S}}}),\ \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\ P'), \Theta'_2$ (by D-$\otimes/\exists$)

Subsubcase: $\Gamma; \Delta'_1 \vdash_\Sigma \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ; P :: (a_{\mathsf{L}} : (\exists B_{\mathsf{S}}.C_{\mathsf{L}}))$,
for some B_{S}, C_{L}, and $(b_{\mathsf{S}} : \hat{B} \in \Gamma)$ such that $A_{\mathsf{L}} = \exists B_{\mathsf{S}}.C_{\mathsf{L}}$ and $\hat{B} \leq B_{\mathsf{S}}$

$\mathsf{proc}(a_{\mathsf{L}},\ \mathsf{send}\ a_{\mathsf{L}}\ b_{\mathsf{S}}\ ; P), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : B_{\mathsf{L}} \multimap C_{\mathsf{L}}, b_{\mathsf{L}} : B_{\mathsf{L}} \vdash_\Sigma \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{L}}\ ; Q :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{L}, C_{L}, and Δ_2 such that $\Delta'_1 = \Delta_2, c_{\mathsf{L}} : B_{\mathsf{L}} \otimes C_{\mathsf{L}}, b_{\mathsf{L}} : B_{\mathsf{L}}$

There exist Θ_2, Θ_2', and $\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta_2'$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\, y_{\mathsf{L}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\, ;P'_{y_{\mathsf{L}}})$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{L}}\, ; Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, y_{\mathsf{L}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\, ; P'_{y_{\mathsf{L}}}), \Theta_2'$
$\quad \longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\, Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, [b_{\mathsf{L}}/y_{\mathsf{L}}]\, P'_{y_{\mathsf{L}}}), \Theta_2'$ (by D-$\multimap$ /Π)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma y_{\mathsf{L}} \leftarrow \mathsf{recv}\, a_{\mathsf{L}}\, ; P_{y_{\mathsf{L}}} :: (a_{\mathsf{L}} : B_{\mathsf{L}} \multimap C_{\mathsf{L}})$,
for some B_{L}, C_{L} such that $A_{\mathsf{L}} = B_{\mathsf{L}} \multimap C_{\mathsf{L}}$

$\mathsf{proc}(a_{\mathsf{L}},\, y_{\mathsf{L}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\, ; P_{y_{\mathsf{L}}}), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : (\Pi B_{\mathsf{S}}.C_{\mathsf{L}}) \vdash_\Sigma \mathsf{send}\, c_{\mathsf{L}}\ b_{\mathsf{S}}\, ; Q :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{S}, C_{L}, $(b_{\mathsf{S}} : \hat{B}) \in \Gamma$, and Δ_2 such that $\hat{B} \leq B_{\mathsf{S}}$ and $\Delta_1' = \Delta_2, c_{\mathsf{L}} : (\Pi B_{\mathsf{S}}.C_{\mathsf{L}})$

There exist Θ_2, Θ_2', and $\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta_2'$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\, y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\, ; P'_{y_{\mathsf{S}}})$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{send}\ c_{\mathsf{L}}\ b_{\mathsf{S}}\, ; Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, y_{\mathsf{S}} \leftarrow \mathsf{recv}\ c_{\mathsf{L}}\, ; P'_{y_{\mathsf{S}}}), \Theta_2'$
$\quad \longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\, Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, [b_{\mathsf{S}}/y_{\mathsf{S}}]\, P'_{y_{\mathsf{S}}}), \Theta_2'$ (by D-$\multimap$ /Π)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma y_{\mathsf{S}} \leftarrow \mathsf{recv}\, a_{\mathsf{L}}\, ; P_{y_{\mathsf{S}}} :: (a_{\mathsf{L}} : (\Pi B_{\mathsf{S}}.C_{\mathsf{L}}))$,
for some B_{S}, C_{L} such that $A_{\mathsf{L}} = \Pi B_{\mathsf{S}}.C_{\mathsf{L}}$

$\mathsf{proc}(a_{\mathsf{L}},\, y_{\mathsf{S}} \leftarrow \mathsf{recv}\ a_{\mathsf{L}}\, ; P_{y_{\mathsf{S}}}), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : \oplus\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma \mathsf{case}\ c_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q} :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{L} and Δ_2 such that $\Delta_1' = \Delta_2, c_{\mathsf{L}} : \oplus\{\overline{l : B_{\mathsf{L}}}\}$

There exist Θ_2, Θ_2', and $\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta_2'$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}},\, c_{\mathsf{L}}.l_h\, ; P')$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\, \mathsf{case}\ c_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow Q}),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, c_{\mathsf{L}}.l_h\, ; P'), \Theta_2'$
$\quad \longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\, Q_h),\, \Theta_2, \mathsf{proc}(a_{\mathsf{L}},\, P'), \Theta_2'$ (by D-$\oplus$)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma a_{\mathsf{L}}.l_h\, ; P :: (a_{\mathsf{L}} : \oplus\{\overline{l : B_{\mathsf{L}}}\})$,
for some B_{L} such that $A_{\mathsf{L}} = \oplus\{\overline{l : B_{\mathsf{L}}}\}$

$\mathsf{proc}(a_{\mathsf{L}},\, a_{\mathsf{L}}.l_h\, ; P), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subsubcase: $\Gamma; \Delta_2, c_{\mathsf{L}} : \&\{\overline{l : B_{\mathsf{L}}}\} \vdash_\Sigma c_{\mathsf{L}}.l_h\, ; Q :: (a_{\mathsf{L}} : A_{\mathsf{L}})$,
for some B_{L} and Δ_2 such that $\Delta_1' = \Delta_2, c_{\mathsf{L}} : \&\{\overline{l : B_{\mathsf{L}}}\}$

There exist Θ_2, Θ_2', and $\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ such that $\Theta_1 = \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}), \Theta_2'$. (by Lemma 10-2)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}})$ is poised (by Definition 3 since Θ_1 is poised)

$\mathsf{proc}(c_{\mathsf{L}},\, P_{c_{\mathsf{L}}}) = \mathsf{proc}(c_{\mathsf{L}}, \mathsf{case}\ c_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P'})$ (by Definition 3 and inversion on process typing)

$\Lambda; \mathsf{proc}(a_{\mathsf{L}},\, c_{\mathsf{L}}.l_h\, ; Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}}, \mathsf{case}\ c_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P'}), \Theta_2'$
$\quad \longrightarrow \Lambda; \mathsf{proc}(a_{\mathsf{L}},\, Q),\, \Theta_2, \mathsf{proc}(c_{\mathsf{L}},\, P'_h), \Theta_2'$ (by D-$\&$)

Subsubcase: $\Gamma; \Delta_1' \vdash_\Sigma \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P} :: (a_{\mathsf{L}} : \&\{\overline{l : B_{\mathsf{L}}}\})$,
for some B_{L} such that $A_{\mathsf{L}} = \&\{\overline{l : B_{\mathsf{L}}}\}$

$\mathsf{proc}(a_{\mathsf{L}}, \mathsf{case}\ a_{\mathsf{L}}\ \mathsf{of}\ \overline{l \Rightarrow P}), \Theta_1$ is poised (by Definition 3 since Θ_1 is poised)

Subcase: Some process in Θ_1 is blocked along a_{S} and $\mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$

Some process in $\mathsf{proc}(a_{\mathsf{L}},\, P_{a_{\mathsf{L}}}),\, \Theta_1$ is blocked along a_{S} and $\mathsf{unavail}(a_{\mathsf{S}}) \in \Lambda$

□

E Examples

E.1 "Imperative" Queue

Linearity prohibits a more "imperative" style of a queue implementation that maintains a reference to the back of the queue for direct insertion of an element. In this section, we give an implementation of a queue with direct access to both the front and back of the queue; the result is shown in Figure 21.

$$\begin{aligned}\mathbf{list}\ \mathbf{A}_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\,\&\{&\mathsf{ins} : \Pi \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{list}\ \mathbf{A}_{\mathsf{S}}\\ &\mathsf{del} : \oplus\{\mathsf{none} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathsf{list}\ \mathbf{A}_{\mathsf{S}},\\ &\qquad\mathsf{some} : \exists \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{list}\ \mathbf{A}_{\mathsf{S}}\}\end{aligned}$$

$$\begin{aligned}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\&\{&\mathsf{enq} : \Pi \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}},\\ &\qquad\mathsf{some} : \exists \mathbf{A}_{\mathsf{S}}.{\downarrow}^{\mathsf{S}}_{\mathsf{L}}\mathbf{queue}\ \mathbf{A}_{\mathsf{S}}\}\}\end{aligned}$$

```
empty : {list A_S}

c ← empty =
  c′ ← accept c ;
  case c′ of
  | ins → x ← recv c′ ;
          e ← empty ;
          c ← detach c′ ;
          c ← elem ← x, e
  | del → c′.none ;
          c ← detach c′ ;
          c ← empty
```

```
elem : {list A_S ← A_S, list A_S}

c ← elem ← x, next =
  c′ ← accept c ;
  case c′ of
  | ins → y ← recv c′ ;
          n ← elem ← y, next ;
          c ← detach c′ ;
          c ← elem ← x, n
  | del → c′.some ;
          send c′ x ;
          c ← detach c′ ;
          fwd c next
```

```
head : {queue A_S ← list A_S, list A_S}

c ← head ← front, back =
  c′ ← accept c ;
  case c′ of
  | enq → x ← recv c′ ;
          back′ ← acquire back ;
          back′.ins ;
          send back′ x ;
          back ← release back′ ;
          c ← detach c′ ;
          c ← head ← front, back
  | deq → front′ ← acquire front ;
          front′.del ;
          (case front′ of
          | none → front ← release front′ ;
                   c′.none ;
                   c ← detach c′ ;
                   c ← head ← front, back
          | some → x ← recv front′ ;
                   front ← release front′ ;
                   c′.some ; send c′ x ;
                   c ← detach c′ ;
                   c ← head ← front, back)
```

Figure 21: Imperative queue: has access to both the front and the end of the queue.

As is common for imperative queue implementations, the queue consists of a head that maintains a reference to the front and the back of a linked list that comprises the elements of the queue. The session type queue A_{S} defines the protocol of the queue, which is the protocol we introduced earlier for the queue in the producer-consumer example. For convenience, we repeat the protocol definition in Figure 21. Process *head* implements this protocol. The session type list A_{S} defines the protocol of the linked list that is used internally by process *head* to store the elements of the queue. Session type list A_{S} is implemented by the processes *empty* and *elem*. Process *empty* represents an empty list, process *elem* represents a non-empty list. The list thus consists of a sequence of *elem* processes, ended by an *empty* process. An empty queue is created by the following lines of code

```
e ← empty ;
q ← head ← e, e ;
```

These lines illustrate that sharing arises in case of an empty queue between the *front* and the *back* channel because they both point to the same list element. For this reason, process *head* defines the channels *front* and *back* to be of the shared session type list A_{S}. Sharing also arises in case of a non-empty queue between a list's *next* channel and the head's *back* channel because both point to the same element in case of the last element in the list. For this reason, process *elem* defines the channel *next* to be of the shared session type list A_{S}. Process *head* uses the acquire-release primitives to communicate with the list. It acquires channel *back* upon receiving an enq request and channel *front* upon receiving a deq request and then releases both channels upon successful insertion into or deletion from the list.

E.2 Cycles and Deadlocks

In the dining philosophers example (Figure 7), the circular dependency among shared channels is created by shared channel process arguments. Alternatively, the circularity can be introduce by shared channel input and output. Figure 22 gives an example that leads to a classic deadlock. The cycle is created by process *client*, which spawns a new shared process *deadlock* and then sends a "self-reference" to that very process after acquiring it. The deadlock occurs once the newly spawned and now linear process attempts to acquire itself.

$$\mathbf{cycle} = \{\uparrow^{\mathsf{S}}_{\mathsf{L}}\Pi\mathbf{cycle}.\mathbf{1}\}$$

```
client : {1}

c ← client =
  d ← deadlock ;
  d′ ← acquire d ;
  send d′ d ;
  wait d′ ;
  close c
```

```
deadlock : {cycle}

c ← deadlock =
  c′ ← accept c ;
  self ← recv c′ ;
  self′ ← acquire self ; (* deadlocks here *)
  send self′ self ;
  wait self′ ;
  close c′
```

Figure 22: Processes creating a cycle along channel *self*, causing a deadlock.

E.3 Cycles and Blocking

Another form of blocking can arise from circularity that is caused by forwarding rather than by a classic deadlock. Figure 23 shows a variation of the previous example where the offering process *terminate* forwards itself to itself, resulting in a configuration in which no process exists anymore along either channel. As a result, the client will block when attempting to re-acquire the process.

$$\mathbf{cycle} = \{\uparrow^{\mathsf{S}}_{\mathsf{L}}\&\{\mathsf{input} : \Pi\mathbf{cycle}.\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathbf{cycle},\ \mathsf{dealloc} : \mathbf{1}\}\}$$

```
client : {1}

c ← client =
  d ← terminate ;
  d′ ← acquire d ;
  d′.input ; send d′ d ;
  d ← release d′ ;
  d′ ← acquire d ; (* blocks here *)
  d′.dealloc ;
  wait d′ ;
  close c
```

```
terminate : {cycle}

c ← terminate =
  c′ ← accept c ;
  case c′ of
  | input → self ← recv c′ ;
            c ← detach c′ ;
            fwd c self
  | dealloc → close c′
```

Figure 23: Client-side blocking caused by circular forwarding.

E.4 Linear Forwarding

In the linear forwarding case of the preservation proof, the following 9 subcases for the types of a_{S} and b_{S} are considered, categorized as shown:

$\hat{B} \leq \hat{A}$:	$a_{\mathsf{S}} : \top$	and	$b_{\mathsf{S}} : \top$
	$a_{\mathsf{S}} : \top$	and	$b_{\mathsf{S}} : \uparrow^{\mathsf{S}}_{\mathsf{L}}B_{\mathsf{L}}$

$$\begin{array}{lllll}
 & a_{\mathsf{S}} : \top & \text{and} & b_{\mathsf{S}} : \bot & \\
 & a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} & \text{and } A_{\mathsf{L}} = B_{\mathsf{L}} \\
 & a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : \bot & \\
 & a_{\mathsf{S}} : \bot & \text{and} & b_{\mathsf{S}} : \bot & \\
\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \neq \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} : & a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} & \text{and } A_{\mathsf{L}} \neq B_{\mathsf{L}} \\
\hat{B} \geq \hat{A} : & a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} & \text{and} & b_{\mathsf{S}} : \top & \\
 & a_{\mathsf{S}} : \bot & \text{and} & b_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}} & \\
 & a_{\mathsf{S}} : \bot & \text{and} & b_{\mathsf{S}} : \bot &
\end{array}$$

Next, we will provide examples for some of the cases above. As in Section 4.2, we use the connective $\supset$ [26, 58] to support value input. The type "int $\supset$ int $\supset$ $\mathbf{1}$", for example, describes as session that receives a value of type int and then continues as a session that expects to receive a value of type int before terminating. As a reminder, the process predicate with the forwarding term is $\mathsf{proc}(a_{\mathsf{L}}, \mathsf{fwd}\ a_{\mathsf{L}}\ b_{\mathsf{L}})$.

In the first 3 cases, $a_{\mathsf{S}} : \top$ guarantees that the channel a_{S} does not yet occur in any process terms in the configuration. In that case, substitution does actually not have any effect. An example for $a_{\mathsf{S}} : \top$ and $b_{\mathsf{S}} : \top$ is:

```
supertype = int ⊃ int ⊃ 1
subtype =   int ⊃ 1

main : {1}
c ← main =
  s ← super ;
  send s 6 ; send s 6 ;
  wait s ; close c

super : {supertype}
a ← super =
  x ← recv a ;
  b ← sub ;
  fwd a b

sub : {subtype}
b ← sub =
  x ← recv b ;
  close b
```

An example for $a_{\mathsf{S}} : \top$ and $b_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$ is:

```
supertype = int ⊃ int ⊃ 1
subtype = ↑SL(int ⊃ 1)

main : {1}
c ← main =
  s ← super ;
  send s 6 ; send s 6 ;
  wait s ; close c

super : {supertype}
a ← super =
  x ← recv a ;
  b ← sub ;
  b′ ← acquire b ;
  fwd a b′

sub : {subtype}
b ← sub =
  b′ ← accept b ;
  x ← recv b′ ;
  close b′
```

An example for $a_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$ and $b_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} B_{\mathsf{L}}$ and $A_{\mathsf{L}} \neq B_{\mathsf{L}}$ is:

```
supertype = ↑SL(int ⊃ int ⊃ 1)
subtype =   ↑SL(int ⊃ 1)

main : {1}
c ← main =
  s ← super ;
  s′ ← acquire s ;
  send s′ 6 ; send s′ 6 ;
  wait s′ ; close c

super : {supertype}
a ← super =
  a′ ← accept a ;
  x ← recv a′ ;
  b ← sub ;
  b′ ← acquire b ;
  fwd a′ b′

sub : {subtype}
b ← sub =
  b′ ← accept b ;
  x ← recv b′ ;
  close b′
```

An example for $a_{\mathsf{S}} : \top$ and $b_{\mathsf{S}} : \bot$ is:

```
supertype = ↑SL(int ⊃ int ⊃ 1)
subtype =   ↑SL(int ⊃ 1)

main : {1}
a ← main =
  b ← super ;
  b′ ← acquire b ;
  send b′ 6 ; send b′ 6 ;
  fwd a b′

super : {supertype}
b ← super =
  b′ ← accept b ;
  x ← recv b′ ;
  s ← sub ;
  s′ ← acquire s ;
  fwd b′ s′

sub : {subtype}
c ← sub =
  c′ ← accept c ;
  x ← recv c′ ;
  close c′
```

Acknowledgements The authors would like to thank Peter Thiemann, Philip Wadler, and other participants for comments on a preliminary talk on this work at the Dagstuhl Seminar on Theory and Applications of Behavioural Types, January 29 – February 3, 2017. The authors would also like to thank Bob Harper for discussions on process calculi and coinduction. This material is based upon work supported by a Mozilla Research grant and partially sponsored by NSF Grant CNS-1423168: "Blameworthy Programs: Accountability via Deviance and Causal Determination". Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Mozilla Research or NSF.

References

[1] R. Arnold. C_0, an imperative programming language for novice computer scientists. Master's thesis, Department of Computer Science, Carnegie Mellon University, Dec. 2010. Available as Technical Report CMU-CS-10-145.

[2] R. Atkey, S. Lindley, and J. G. Morris. Conflation confers concurrency. In S. L. et al., editor, *Wadler Festschrift*, pages 32–55. Springer LNCS 9600, 2016.

[3] R. Beauxis, C. Palamidessi, and F. D. Valencia. On the asynchronous nature of the asynchronous π-calculus. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 473–492. Springer, 2008.

[4] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *8th International Workshop on Computer Science Logic (CSL)*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 1994. An extended version appeared as Technical Report UCAM-CL-TR-352, University of Cambridge.

[5] K. Bierhoff and J. Aldrich. Modular typestate checking of aliased objects. In *22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'07)*, pages 301–320. ACM, 2007.

[6] J. Boyland. Checking interference with fractional permissions. In *10th International Symposium on Static Analysis (SAS)*, pages 55–72, 2003.

[7] S. D. Brookes. A semantics for concurrent separation logic. In *15th International Conference on Concurrency Theory (CONCUR)*, volume 3170 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2004.

[8] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *21st International Conference on Concurrency Theory (CONCUR)*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010.

[9] L. Caires, F. Pfenning, and B. Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016.

[10] E. Castegren and T. Wrigstad. Lolcat: Relaxed linear references for lock-free programming. Technical Report 2016-013, Uppsala University, July 2016.

[11] I. Cervesato and A. Scedrov. Relating state-based and process-based concurrency through linear logic. *Information and Computation*, 207(10):1044–1077, 2009.

[12] I. Cervesato, F. Pfenning, D. Walker, and K. Watkins. A concurrent logical framework ii: Examples and applications. Technical Report CMU-CS-02-102, Computer Science Department, Carnegie Mellon University, 2002. Revised May 2003.

[13] B.-Y. E. Chang, K. Chaudhuri, and F. Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, School of Computer Science, Carnegie Mellon University, December 2003.

[14] K. Crary, R. Harper, and S. Puri. What is a recursive module? In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 50–63, 1999.

[15] R. DeLine and M. Fähndrich. Typestates for objects. In *18th European Conference on Object-Oriented Programming (ECOOP)*, volume 3086 of *Lecture Notes in Computer Science*, pages 465–490. Springer, 2004.

[16] Y. Deng, R. J. Simmons, and I. Cervesato. Relating reasoning methodologies in linear logic and process algebra. *Mathematical Structure in Computer Science*, 26(5):868–906, Jan. 2016.

[17] H. DeYoung, L. Caires, F. Pfenning, and B. Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In P. Cégielski and A. Durand, editors, *Proceedings of the 21st Conference on Computer Science Logic*, CSL 2012, pages 228–242, Fontainebleau, France, Sept. 2012. Leibniz International Proceedings in Informatics.

[18] M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session types for object-oriented languages. In *20th European Conference on Object-Oriented Programming (ECOOP)*, volume 4067 of *Lecture Notes in Computer Science*, pages 328–352. Springer, 2006.

[19] E. W. Dijkstra. Hierarchical ordering of sequential processes. EWD Manuscript 310, 1971–1973.

[20] M. Fähndrich and R. DeLine. Adoption and focus: Practical linear types for imperative programming. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 13–24. ACM, 2002.

[21] C. Flanagan and S. Qadeer. A type and effect system for atomicity. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 338–349. ACM, 2003.

[22] S. J. Gay and M. Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2–3):191–225, 2005.

[23] D. K. Gifford and J. M. Lucassen. Integrating functional and imperative programming. In *LISP and Functional Programming*, pages 28–38, 1986.

[24] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[25] D. Griffith. *Polarized Substructural Session Types*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[26] D. Griffith and F. Pfenning. SILL. `https://github.com/ISANobody/sill`, 2015.

[27] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2013.

[28] S. Heule, K. R. M. Leino, P. Müller, and A. J. Summers. Abstract read permissions: Fractional permissions without the fractions. In *14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 7737 of *Lecture Notes in Computer Science*, pages 315–334. Springer, 2013.

[29] K. Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory (CONCUR)*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[30] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *7th European Symposium on Programming (ESOP)*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[31] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 273–284. ACM, 2008.

[32] W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press (1980), 1969.

[33] R. Hu, N. Yoshida, and K. Honda. Session-based distributed programming in Java. In *22nd European Conference on Object-Oriented Programming (ECOOP)*, volume 5142 of *Lecture Notes in Computer Science*, pages 516–541. Springer, 2008.

[34] T. B. L. Jespersen, P. Munksgaard, and K. F. Larsen. Session types for Rust. In *11th ACM SIGPLAN Workshop on Generic Programming (WGP)*, 2015.

[35] L. Jia, H. Gommerstadt, and F. Pfenning. Monitors and blame assignment for higher-orderrder session types. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 582–594, 2016.

[36] R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 637–650. ACM, 2015.

[37] N. Kobayashi and C. Laneve. Deadlock analysis of unbounded process networks. *Information and Computation*, 252:48–70, 2017.

[38] J. Lange, N. Ng, B. Toninho, and N. Yoshida. Fencing off go: Liveness and safety for channel-based programming. In *44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2017. To appear.

[39] K. R. M. Leino and P. Müller. A basis for verifying multi-threaded programs. In *18th European Symposium on Programming (ESOP)*, pages 378–393, 2009.

[40] F. Militão, J. Aldrich, and L. Caires. Rely-guarantee protocols. In *28th European Conference on Object-Oriented Programming (ECOOP)*, volume 8586 of *Lecture Notes in Computer Science*, pages 334–359. Springer, 2014.

[41] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[42] MozillaResearch. The Rust programming language. `https://doc.rust-lang.org/stable/book`, November 2016.

[43] K. Naden, R. Bocchino, J. Aldrich, and K. Bierhoff. A type system for borrowing permissions. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 557–570. ACM, 2012.

[44] R. Neykova and N. Yoshida. Multiparty session actors. In *16th International Conference on Coordination Models and Languages (COORDINATION)*, volume 8459 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2014.

[45] L. Nistor, J. Aldrich, S. Balzer, and H. Mehnert. Object propositions. In *19th International Symposium on Formal Methods (FM'14)*, Lecture Notes in Computer Science. Springer, 2014. To appear.

[46] P. W. O'Hearn. Resources, concurrency and local reasoning. In *15th International Conference on Concurrency Theory (CONCUR)*, volume 3170 of *Lecture Notes in Computer Science*, pages 49–67. Springer, 2004.

[47] J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014.

[48] F. Pfenning. C0 language. http://c0.typesafety.net, 2010.

[49] F. Pfenning and D. Griffith. Polarized substructural session types. In *18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 9034 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[50] F. Pfenning, T. J. Cortina, and W. Lovas. Teaching imperative programming with contracts at the freshmen level. Unpublished note, 2011.

[51] J. Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, January 2009. URL `http://www.cs.cmu.edu/~jcreed/papers/jdml.pdf`.

[52] D. Sangiorgi and D. Walker. *The π-Calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.

[53] A. Scalas and N. Yoshida. Lightweight session programming in Scala. In *30th European Conference on Object-Oriented Programming (ECOOP)*, volume 56 of *LIPIcs*, pages 21:1–21:28. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[54] J. Smans, B. Jacobs, and F. Piessens. Implicit dynamic frames: Combining dynamic frames and separation logic. In *23rd European Conference on Object-Oriented Programming (ECOOP'09)*, volume 5653 of *Lecture Notes in Computer Science*, pages 148–172. Springer, 2009.

[55] F. Smith, D. Walker, and J. G. Morrisett. Alias types. In *9th European Symposium on Programming (ESOP)*, pages 366–381, 2000.

[56] R. E. Strom and S. Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering (TSE)*, 12(1):157–171, 1986.

[57] B. Toninho. *A Logical Foundation for Session-based Concurrent Computation*. PhD thesis, Carnegie Mellon University and New University of Lisbon, 2015.

[58] B. Toninho, L. Caires, and F. Pfenning. Higher-order processes, functions, and sessions: a monadic integration. In *22nd European Symposium on Programming (ESOP)*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013.

[59] A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency. In *18th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 377–390. ACM, 2013.

[60] V. Vafeiadis. Concurrent separation logic and operational semantics. *Electronic Notes in Theoretical Computer Science*, 276:335–351, 2011.

[61] P. Wadler. Linear types can change the world! In *Woking Conference on Programming Concepts and Methods*, 1990.

[62] P. Wadler. Propositions as sessions. In *17th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 273–286. ACM, 2012.

[63] D. Walker and K. Watkins. On regions and linear types. In *6th ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*, pages 181–192. ACM, 2001.

[64] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework i: Judgments and properties. Technical Report CMU-CS-02-101, Computer Science Department, Carnegie Mellon University, 2002. Revised May 2003.

[65] M. Willsey, R. Prabhu, and F. Pfenning. Design and implementation of Concurrent C0. In *Fourth International Workshop on Linearity*, June 2016.