arXiv:2101.06249v1 [cs.PL] 15 Jan 2021

Manifestly Phased Communication via Shared Session Types

Chuta Sano, Stephanie Balzer, and Frank Pfenning

January 18, 2021

Abstract

Session types denote message protocols between concurrent processes, allowing a type-safe formalization of inter-process communication. Although previous works demonstrate a well-defined notion of subtyping where processes have different perceptions of the protocol, these formulations were limited to linear session types where each channel of communication has a unique provider and client. In our work, we extend these previous formulations into the shared session type setting where channels can now have multiple clients instead of a single client. We demonstrate that this allows shared sessions to be released at a different type, allowing the encoding of phases in a shared protocol to be manifest in the session type.

Contents

1 Introduction

Session types prescribe bidirectional communication protocols between concurrent processes [Hon93; HVK98]. Variations of this type system were later given logical correspondences to *intuitionistic* [CP10] and *classical* [Wad12] linear logic where proofs correspond to programs and cut reduction to communication.

The correspondence to linear logic mainly provided an interpretation of *linear session types*, which denote sessions with exactly one client and one provider. A *shared session type*, with multiple clients and one provider, was given a *copying semantics* interpretation from the exponential operator in linear logic, where processes providing a shared channel effectively duplicated itself for every client. In particular, under this interpretation, a server providing a shared channel to n clients would create a total of n independent copies of itself that communicate with the n clients linearly.

Unfortunately, the copying semantics cannot model scenarios that demand sharing of resources since clients receive their own unique provider. To model these scenarios, prior work [BP17] proposed a *sharing semantics* by stratifying session types into shared and linear modalities and decomposing the aforementioned exponential operator into modal shifts between the two modalities. Although we believe that both copying and sharing semantics are compatible with each other in the sense that they can co-exist in the same system, in this paper, we assume "shared" to refer to the sharing semantics approach as opposed to copying semantics unless explicitly stated.

Communication adhering to shared session types follow an *acquire-release* discipline where a client first *acquires* exclusive access to the provider, continues linearly, and then finally *releases* the exclusive access, allowing the channel to be acquired by another client. One of the key requirements to guarantee soundness in that system is that processes must be released[1] at the same type that it was acquired since there could be multiple clients waiting to acquire a particular shared channel. Prior work [BP17] demonstrated that this requirement, or *equi-synchronizing constraint* can be formulated from the provider's viewpoint as a static constraint on the session type.

In a previous work [San19], we identified many practical scenarios, which we highlight in Section 4, where clients using a shared channel often follow particular *phases* across successive acquire-release cycles. However, we noted that this cannot be expressed directly in the type system since the equi-synchronizing constraint requires that types remain consistent across acquire-release cycles.

[1]We also allow termination vacuously

In this paper, we introduce a formulation of subtyping compatible with shared session types. In this system, we statically relax the constraint that clients and providers must have the same view of a channel's type and instead allow clients to locally view a channel's type as a supertype of the channel type that is provided. We also generalize the equi-synchronizing constraint, taking into account the subtle consequences that emerge due to the disagreement of the type of a shared channel between clients and the provider. This gives us a rich session type system that allows manifestation of phases in communication directly in the type system.

The main contributions of this paper include:

- a full formalization of a subtyping relation for shared session types and their meta theory
- the formulation of the safely synchronizing constraint, which is a relaxation of the equi-synchronizing constraint that takes advantage of subtyping
- a detailed description of $SILL_{S\leq}$, which incorporates the subtyping that we propose, consisting of its type system, synchronous operational semantics, and meta theory including the usual preservation and progress proofs

The rest of the paper will proceed as follows:

- In Section 2 we give a brief introduction to both linear and shared session-typed message-passing concurrency.
- In Section 3 we present our system along with the preservation and progress theorem statements.
- In Section 4 we provide examples of scenarios that take advantage of the new subtyping system.
- In Section 5 we outline related works and their connections to our paper.
- In Section 6 we conclude the main part of the paper with discussions of limitations, further generalizations, and some conjectures.
- In the Appendix we provide a formal presentation of our system along with the technical proofs of relevant theorems and lemmas, notably the preservation F and modified progress G theorems.

2 Background

2.1 Linear Session Types

The core observation behind the theory of session types is that the (intuitionistic) linear sequent

$$A_1, A_2, \ldots, A_n \vdash B$$

can be viewed as a typing judgment for a process P by annotating the linear propositions with channel names:

$$a_1 : A_1, a_2 : A_2, \ldots, a_n : A_n \vdash P :: b : B$$

Here we say that process P *provides* a session of type B across channel b while *using* channels $a_1, \ldots, a_n$ with session types $A_1, \ldots, A_n$ respectively. We say that P provides b and is a client to $a_1, \ldots, a_n$.

Since the session type associated with a channel denotes a bidirectional protocol, each type connective has two operational interpretations[2] – one from the perspective of a provider and one from the client. For example, a channel of type $A \otimes B$ requires that the provider send a channel of type A and proceed as B while the client receive a channel of type A and proceed as A. As shown in Table 1, this operationally dual interpretation culminates in a style where for any connective, either the client or provider will send while the other will receive.

Type	Operational interpretation from provider	Operational interpretation from client
1	End of protocol – close channel	Wait for the provider to close the channel
$A \otimes B$	Send channel of type A and proceed as B	Receive channel of type A and proceed as B
$A \multimap B$	Receive channel of type A and proceed as B	Send channel of type A and proceed as B
$\oplus\{\overline{l : A}\}$	Send a label $i \in \bar{l}$ and proceed as A_i	Receive and branch on $i \in \bar{l}$ and proceed as A_i
$\&\{\overline{l : A}\}$	Receive and branch on $i \in \bar{l}$ and proceed as A_i	Send a label $i \in \bar{l}$ and proceed as A_i

Table 1: A summary of the linear connectives and their operational interpretations

Another point we would like to emphasize is that all connectives follow a pattern where they send or receive and then proceed as some type A. This type A is referred to as the *continuation type*.

Example Taking int as a primitive[3], we can denote a session type for a simple server that performs arithmetic operations and communicates back the result. The protocol should begin with the client choosing an operation – *add* or *sub*, then for both branches, sending two ints, and then finally receiving an int. This can be expressed as:

$$\begin{aligned} \text{arith} = \&\{&\mathit{add} : \text{int} \multimap \text{int} \multimap \text{int} \otimes 1, \\ &\mathit{sub} : \text{int} \multimap \text{int} \multimap \text{int} \otimes 1\} \end{aligned}$$

[2]This is due to the privileged position of the right side of the sequent in intuitionistic logic.

[3]It is important to note that int is a session type and therefore is encoded as a channel. For simplicity, we will treat it as a channel carrying an integer.

A process using a channel of this type must first send a label *add* or *sub*. After sending a label, the client proceeds to use the channel which is now of type $\text{int} \multimap \text{int} \multimap \text{int} \otimes 1$ (both labels lead to the same continuation type). The client must then send a channel of type int and proceed as $\text{int} \multimap \text{int} \otimes 1$. Again, the client must send another channel of type int and proceed as $\text{int} \otimes 1$. This time, the client must receive an int and proceed as 1. At this point, the client must wait for the session to terminate. The operational interpretation from the provider's point of view is similar, with sending and receiving flipped. For example, the provider must first receive a label, and so on.

To emphasize that the different branches from the external choice & can be different, we will add a negation operation, which require the client to send one int (instead of two) and then receive the result:

$$\begin{aligned} \text{arith} = \&\{&\mathit{add} : \text{int} \multimap \text{int} \multimap \text{int} \otimes 1, \\ &\mathit{sub} : \text{int} \multimap \text{int} \multimap \text{int} \otimes 1, \\ &\mathit{neg} : \text{int} \multimap \text{int} \otimes 1\} \end{aligned}$$

Currently, we can only terminate a session through the unit type 1, but for this example, it might make more sense for the process to recurse, allowing clients to perform as many operations as it desires. We adopt an *equi-recursive* [CHP99] interpretation which require that recursive session types be *contractive* [GH05], guaranteeing that there are no messages associated with the unfolding of a recursive type. Adding recursion to our ongoing example, we have

$$\begin{aligned} \text{arith} = \&\{&\mathit{add} : \text{int} \multimap \text{int} \multimap \text{int} \otimes \text{arith}, \\ &\mathit{sub} : \text{int} \multimap \text{int} \multimap \text{int} \otimes \text{arith}, \\ &\mathit{neg} : \text{int} \multimap \text{int} \otimes \text{arith}\} \end{aligned}$$

Readers familiar with linear logic might notice that the linear session type arith will never terminate, which means that any clients using the channel can also never terminate. For completeness sake, we will provide one final modification to safely allow termination by adding an "exit" label to the external choice.

$$\begin{aligned} \text{arith} = \&\{&\mathit{add} : \text{int} \multimap \text{int} \multimap \text{int} \otimes \text{arith}, \\ &\mathit{sub} : \text{int} \multimap \text{int} \multimap \text{int} \otimes \text{arith}, \\ &\mathit{neg} : \text{int} \multimap \text{int} \otimes \text{arith}, \\ &\mathit{exit} : 1\} \end{aligned}$$

2.1.1 Type judgments

We say that a linear process term P offers a channel z of type C by using client channels $\Delta = x_1 \colon A_1, \ldots, x_n \colon A_n$. This is expressed as a judgment:

$$\Delta \vdash P :: (z \colon C)$$

These judgments have direct correspondence with the sequent calculus presentation of dual intuitionistic linear logic, where process terms are given intuitive names following a C-like syntax.

$$\frac{\Delta \vdash P :: (z \colon C)}{\Delta, x \colon 1 \vdash \mathsf{wait}\ x; P :: (z \colon C)}\ 1L \qquad \frac{}{\cdot \vdash \mathsf{close}\ x :: (x \colon 1)}\ 1R$$

$$\frac{\Delta, x \colon B, y \colon A \vdash P :: (z \colon C)}{\Delta, x \colon A \otimes B \vdash y \leftarrow \mathsf{recv}\ x; P :: (z \colon C)}\ \otimes L \qquad \frac{\Delta \vdash P :: (x \colon B)}{\Delta, y \colon A \vdash \mathsf{send}\ x\ y; P :: (x \colon A \otimes B)}\ \otimes R$$

where for example the process term in $1L$, wait $x; P$ denotes a process that waits for channel x to terminate and then continue executing the remaining statement P.

One important point is that contraction and weakening are rejected. In particular, this means that linear channels that a process is using cannot be duplicated and must be fully consumed before terminating as shown in $1R$. We give a more complete presentation of the type judgments in Section 2.2.1 after introducing shared session types.

2.1.2 Configuration

Since process calculi consist of many concurrent processes that interact with each other, we reason about not just one process calculi term but a collection of terms by introducing a process configuration, or a collection of linear process terms. For the purely linear setting, a configuration can be viewed as a forest where the parent are clients to its children.

Figure 1 demonstrates a simple configuration for linear process terms. In the paper, we represent the configuration as a list of processes with an ordering constraint that processes using channels provided by other processes must appear before those processes – for example, the configuration matching the one in the figure can be written as a list P_1, Q_1, Q_2, P_2. There are many valid permutations to this list, since the only constraint we have is that P_1 appear before Q_1 and Q_2 because P_1 uses channels provided by Q_1 and Q_2 (a_1 and a_2 respectively). We will later expand on this in Section 2.2.2.

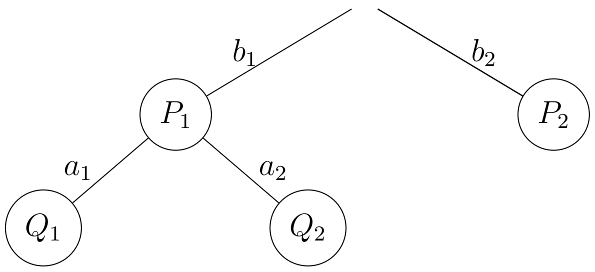

Figure 1: A visualization of a linear configuration consisting of four process terms. The top level configuration processes P_1 and P_2 collectively provide channels b_1 and b_2 (represented by an edge to an absent root node), while the channels a_1 and a_2 provided by Q_1 and Q_2 respectively are used by P_1. Thus, we say that P_1 is a client to a_1 and a_2.

A configuration is well-typed if each process is well-typed according to the previously introduced type judgments. For this particular example, we require that

$$\begin{aligned} \cdot \vdash Q_1 &:: (a_1 \colon A_1) \\ \cdot \vdash Q_2 &:: (a_2 \colon A_2) \\ a_1 \colon A_1, a_2 \colon A_2 \vdash P_1 &:: (b_1 \colon B_1) \\ \cdot \vdash P_2 &:: (b_2 \colon B_2) \end{aligned}$$

for some linear session types A_1, A_2, B_1, and B_2.

2.2 Shared Session Types

Although linear session types and their corresponding process calculi provide strong guarantees such as *session fidelity* (preservation) and *deadlock freedom* (progress), they are not expressive enough to model systems with shared resources. Following prior work [BP17], we extend the system to support shared channels, which connect a single provider with multiple clients, and stratify session types into shared and linear types. In the remaining sections, we will make this distinction explicit by marking channel names and session type meta-variables with the subscripts S and L respectively where appropriate. For example, a linear channel is marked a_L, while a shared channel is marked b_S.

We next introduce modal shift operators that allow session types to transition from linear to shared ($\uparrow_L^S$) and from shared to linear ($\downarrow_L^S$). In summary, we have:

$$\begin{aligned} A_S &::= \uparrow_L^S A_L \\ A_L, B_L &::= \downarrow_L^S A_S \mid 1 \mid A_L \otimes B_L \mid A_L \multimap B_L \mid \&\{\overline{l{:}A_L}\} \mid \oplus\{\overline{l{:}A_L}\} \end{aligned}$$

where we point out that the previously defined (linear) type operators such as $\otimes$ remain only at the linear layer – a shared session type can only be constructed by a modal upshift $\uparrow_L^S$ of some linear session type A_L.

What remains are the operational interpretations of the modal shifts $\uparrow_L^S$ and $\downarrow_L^S$. From the client's perspective, encountering a channel of type $\uparrow_L^S A_L$ means that they must acquire exclusive access of the channel and then continue the session as specified by A_L. From the provider's perspective, $\uparrow_L^S A_L$ means that it must accept an acquire request by a client and then transition to a linear process, now offering a linear channel A_L. On the other hand, when encountering $\downarrow_L^S A_S$, the client must release the exclusive access it had and then continue the session as A_S whereas the provider must detach from its linear state and transition back to a shared process. This is summarized in Table 2.

Type	Operational interpretation from provider	Operational interpretation from client
1	End of protocol – close channel	Wait for the provider to close the channel
$A_L \otimes B_L$	Send channel of type A_L and proceed as B_L	Receive channel of type A_L and proceed as B_L
$A_L \multimap B_L$	Receive channel of type A_L and proceed as B_L	Send channel of type A_L and proceed as B_L
$\oplus\{\overline{l{:}A_L}\}$	Send a label $i \in \overline{l}$ and proceed as A_{i_L}	Receive and branch on $i \in \overline{l}$ and proceed as A_{i_L}
$\&\{\overline{l{:}A_L}\}$	Receive and branch on $i \in \overline{l}$ and proceed as A_{i_L}	Send a label $i \in \overline{l}$ and proceed as A_{i_L}
$\downarrow_L^S A_S$	Detach from a linear session and proceed as A_S	Release exclusive access and proceed as A_S
$\uparrow_L^S A_L$	Accept an acquire request and proceed as A_L	Acquire and proceed as A_L

Table 2: An extension to Table 1, adding in the new modality distinctions and two new type operators $\uparrow_L^S$ and $\downarrow_L^S$

One of the key observations with this addition is that clients of shared channels generally follow an acquire-release pattern – they must first acquire exclusive access to the channel, then proceed linearly, and then finally release the exclusive access that they had, allowing other clients of the same shared channel to potentially acquire exclusive access.

There remains one obstacle to session fidelity with shared session types as we have formulated thus far; consider for example a shared session type such as $A_S = \uparrow_L^S \&\{a : \downarrow_L^S A_S, b : \downarrow_L^S B_S\}$ for some unrelated B_S. Suppose there are two clients M and N to some $a_S{:}A_S$, where M is blocked waiting to acquire a_S while N successfully acquires a_S. Since N transitions linearly according to the protocol, it must send one of two labels a or b due to the external choice. If N sends the label b, then it must release a_S to some unrelated type B_S. Now M, who thinks it is trying to acquire $a_S{:}A_S$ will instead acquire $a_S{:}B_S$, violating session fidelity.

Previous work [BP17] addresses this problem by adding an additional constraint that if a channel was acquired at some type C_S, all possible future releases (by looking at the continuation types) must release at C_S. This is formulated as an equi-synchronizing constraint on session types directly. For example, $C_S = \uparrow_L^S \& \{a : \downarrow_L^S C_S, b : 1 \otimes \downarrow_L^S C_S\}$ would be an equi-synchronizing type because it releases at C_S in all possible continuations whereas the previously defined A_S would not because not all continuations (the b branch) lead to releasing at A_S.

2.2.1 Type judgments

Type judgments of a process are now of form:

$$\Gamma; \Delta \vdash P :: (z_L{:}C_L)$$
$$\Gamma \vdash Q :: (z_S{:}C_S)$$

The former is read as "linear process term P offers a (linear) channel z_L of type C_L using shared channels Γ and linear channels Δ." The latter is read as "shared process term Q offers a (shared) channel z_S of type C_S using shared channels Γ," where there must be no dependency on any linear channels due to the independence principle [BP17].

Global signature In the following sections, we will implicitly assume a global signature Σ, which is a set of process definitions that can be thought as the process calculi analogue to a signature consisting of function definitions. A process definition consists of the offering channel name and its type, the client channel names and their types, and the process term:

$$x_L{:}A_L \leftarrow X_L \leftarrow \overline{y_L{:}B_L}, \overline{w_S{:}E_S} = P$$
$$z_S{:}C_S \leftarrow Z_L \leftarrow \overline{v_S{:}D_S} = Q$$

The former denotes a linear process definition of a process named X_L that offers a channel $x_L{:}A_L$ while using linear channels $y_{1L}{:}B_{1L}, \ldots, y_{nL}{:}B_{nL}$ and shared channels $w_{1S}{:}E_{1S}, \ldots, w_{mS}{:}E_{mS}$ for some n and m, where P consists of its implementation. Similarly, the latter denotes a shared process definition of a process named Z_S that offers a channel $z_S{:}C_S$ while using shared channels $v_{1S}{:}D_{1S}, \ldots, v_{nS}{:}D_{nS}$ for some n, where Q consists of its implementation. Again, it is important that shared process definitions do not depend on linear channels due to the independence principle.

We complete the type judgments for the purely linear operators that we introduced previously.

$$\frac{}{\Gamma; y_L{:}A_L \vdash \text{fwd } x_L\ y_L :: (x_L{:}A_L)}\ ID_L$$

$$\frac{\overline{w_S{:}E_S} \in \Gamma \quad \big(x'_L{:}A_L \leftarrow X_L \leftarrow \overline{y'_L{:}B_L}, \overline{w'_S{:}E_S} = P\big) \in \Sigma \quad \Gamma; \Delta, x_L{:}A_L \vdash Q :: (z_L{:}C_L)}{\Gamma; \Delta, \overline{y_L{:}B_L} \vdash x_L \leftarrow X_L \leftarrow \overline{y_L}, \overline{w_S}; Q :: (z_L{:}C_L)}\ SP_{LL}$$

$$\frac{\Gamma; \Delta \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}1 \vdash \text{wait } x_L; P :: (z_L{:}C_L)}\ 1L \qquad \frac{}{\Gamma; \cdot \vdash \text{close } x_L :: (x_L{:}1)}\ 1R$$

$$\frac{\Gamma; \Delta, x_L{:}B_L, y_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}A_L \otimes B_L \vdash y_L \leftarrow \text{recv } x_L; P :: (z_L{:}C_L)}\ \otimes L \qquad \frac{\Gamma; \Delta \vdash P :: (x_L{:}B_L)}{\Gamma; \Delta, y_L{:}A_L \vdash \text{send } x_L\ y_L; P :: (x_L{:}A_L \otimes B_L)}\ \otimes R$$

$$\frac{\Gamma; \Delta, x_L{:}B \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}A_L \multimap B_L, y_L{:}A_L \vdash \text{send } x_L\ y_L; P :: (z_L{:}C_L)}\ \multimap L \qquad \frac{\Gamma; \Delta, y_L{:}A_L \vdash P :: (x_L{:}B_L)}{\Gamma; \Delta \vdash y_L \leftarrow \text{recv } x_L; P :: (x_L{:}A_L \multimap B_L)}\ \multimap R$$

$$\frac{\forall i \in \bar{l} \quad \Gamma; \Delta, x_L{:}A_{iL} \vdash P_i :: (c_L{:}Z_L)}{\Gamma; \Delta, x_L{:} \oplus \{\overline{l{:}A_L}\} \vdash \text{case } x_L \text{ of } \{\overline{l \Rightarrow P}\} :: (c_L{:}Z_L)}\ \oplus L \qquad \frac{i \in \bar{l} \quad \Gamma; \Delta \vdash P :: (x_L{:}A_{iL})}{\Gamma; \Delta \vdash x.i; P :: (x_L{:} \oplus \{\overline{l{:}A_L}\})}\ \oplus R$$

$$\frac{i \in \bar{l} \quad \Gamma; \Delta, x_L{:}A_{iL} \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}\&\{\overline{l{:}A_L}\} \vdash x.i; P :: (z_L{:}C_L)}\ \& L \qquad \frac{\forall i \in \bar{l} \quad \Gamma; \Delta \vdash P_i :: (x_L{:}A_{iL})}{\Gamma; \Delta \vdash \text{case } x_L \text{ of } \{\overline{l \Rightarrow P}\} :: (x_L{:}\&\{\overline{l{:}A_L}\})}\ \& R$$

where structural rules, consisting of exchange, contraction, and weakening of Γ and exchange of Δ are taken to be implicit.

Adopting the modalities give rise to seven additional typing rules; an ID for shared processes, spawning a shared channel from a linear process, spawning a shared channel from a shared process, and four judgments for the two new shift operators with their left and right rules:

$$\frac{}{\Gamma, y_S B \vdash \text{fwd } x_S\ y_S :: (x_S{:}A_S)}\ ID_S$$

$$\frac{\overline{y_S{:}B_S} \in \Gamma \quad \big(x'_S{:}A_S \leftarrow X_S \leftarrow \overline{y'_S{:}B_S} = P\big) \in \Sigma \quad \Gamma, x_S{:}A_S; \Delta \vdash Q :: (z_L{:}C_L)}{\Gamma; \Delta \vdash x_S \leftarrow X_S \leftarrow \overline{y_S}; Q :: (z_L{:}C_L)}\ SP_{LS}$$

$$\frac{\overline{y_S} B \in \Gamma \quad \big(x'_S{:}A_S \leftarrow X_S \leftarrow \overline{y'_S{:}B'_S} = P\big) \in \Sigma \quad \Gamma, x_S{:}A_S \vdash Q :: (z_S{:}C_S)}{\Gamma \vdash x_S \leftarrow X_S \leftarrow \overline{y_S}; Q :: (z_S{:}C_S)}\ SP_{SS}$$

$$\frac{\Gamma, x_S A; \Delta, x_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma, x_S A; \Delta \vdash x_L \leftarrow \text{acq}_S\ x_S; P :: (z_L{:}C_L)}\ {\uparrow}^S_L L \qquad \frac{\Gamma; \cdot \vdash P :: (x_L{:}A_L)}{\Gamma \vdash x_L \leftarrow \text{acc}_S\ x_S; P :: (x_S{:}{\uparrow}^S_L A_L)}\ {\uparrow}^S_L R$$

$$\frac{\Gamma, x_S{:}A_S; \Delta \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}\downarrow_L^S A_S \vdash x_S \leftarrow \mathrm{rel}_S\ x_S; P :: (z_L{:}C_L)} \downarrow_L^S L \quad \frac{\Gamma \vdash P :: (x_S{:}A_S)}{\Gamma; \cdot \vdash x_S \leftarrow \mathrm{det}_S\ x_S; P :: (x_L{:}\downarrow_L^S A_S)} \downarrow_L^S R$$

We will later modify this system to take into account the safely synchronizing constraint and the extended type system in Section 3.

2.2.2 Configuration

We similarly stratify the configuration Ω into a shared fragment Λ and a linear fragment Θ. A shared configuration consists of a collection of shared processes $\mathrm{proc}(a_S, P)$ and placeholders for unavailable channels (for example, due to it currently being acquired and in the linear state) $\mathrm{unavail}(a_S)$. Following this pattern, we also update our previous formulation of the linear configuration Θ to a list of linear processes $\mathrm{proc}(a_L, P)$. The first argument in the proc terms denotes the channel name that the process is providing. For visualization purposes, it is helpful to think of the shared configuration to be free graphs and the linear configuration to be forests.

As demonstrated in Figure 2, the linear fragment of the configuration still maintains the linearity constraint – there is only one arrow to a given linear process with a strict forest-ordering. However, the shared fragment of the configuration loosely allow any dependencies – as showcased, multiple processes can be clients to S_1, and a shared process can even be a client to itself as shown in S_2[4]. If P_1 successfully acquires d_S, then $\mathrm{proc}(d_S, S_1)$ transitions to $\mathrm{unavail}(d_S)$ and a $\mathrm{proc}(d_L, -)$ with the appropriate process term will appear as a child to P_1.

3 Subtyping

3.1 Linear Session Types

Taking nat to be a primitive compatible with int[5], we consider a (linear) session type $\mathrm{nat} \otimes A_L$. According to this protocol, the provider must first send a channel of type nat, which we informally take to be primitive for the sake of this example. In previous works, the client must act according to the same protocol; they must receive a channel of type nat. However, it seems reasonable for a client to instead act according to a slightly different protocol. Instead of receiving a channel of type nat, they can receive a channel of type int, which we again take as a primitive, since anything of type nat can be interpreted as an int. Thus, the client can safely interpret this channel's protocol to be a session of form $\mathrm{int} \otimes A_L$.

[4]Although there is nothing computationally interesting about it – S_2 will deadlock if it tries to acquire itself

[5]For example, let $\mathrm{nat} = \oplus\{pos : num\}$, $\mathrm{int} = \oplus\{pos : num, neg : num\}$, and $\mathrm{num} = \oplus\{s : num, z : 1\}$

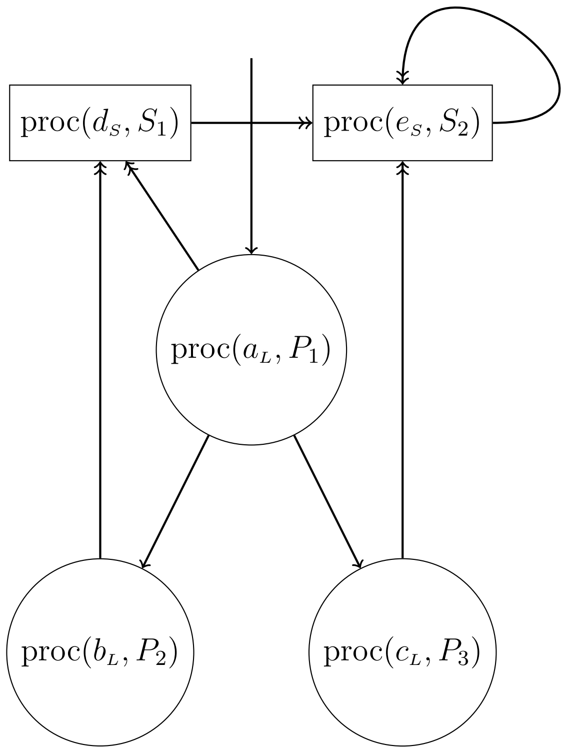

Figure 2: A visualization of a configuration consisting of linear processes P_1, P_2, and P_3, represented as circles, and shared processes S_1 and S_2, represented as rectangles. The direction of the arrow indicate that the destination is the provider; for example, P_1 uses channels provided by P_2, P_3, and S_1. The arrow to P_1 indicates that P_1 is a top level process term.

On the other hand, consider a session type int $\multimap A_L$. The provider must receive a channel of type int while the client must send a channel of type int. In this example, the client can instead send a channel of type nat. Therefore, the client can safely interpret the session to be of type nat $\multimap A_L$.

Now consider a session type $\&\{a : A_L, b : B_L\}$. Since the provider is waiting for one of two choices, a or b, the client can take the type to be a subset of the available choices, for example, $\&\{b : B_L\}$. Similarly, for $\oplus\{a : A_L, b : B_L\}$, since the provider makes a decision between a and b, a client can wait on additional choices, for example, $\oplus\{a : A_L, b : B_L, c : C_L, d : D_L\}$.

In general, if $A \leq B$, then a provider viewing its offering channel as A can safely communicate with a client viewing the channel as B. The following subtyping rules, interpreted coinductively, generalize the examples noted above.

$$\frac{}{1 \leq 1} \leq_1 \qquad \frac{A_L \leq A'_L \qquad B_L \leq B'_L}{A_L \otimes B_L \leq A'_L \otimes B'_L} \leq_\otimes \qquad \frac{A'_L \leq A_L \qquad B_L \leq B'_L}{A_L \multimap B_L \leq A'_L \multimap B'_L} \leq_\multimap$$

$$\frac{\forall i \in \overline{l} \quad A_{i_L} \leq A'_{i_L}}{\oplus\{\overline{l{:}A_L}\} \leq \oplus\{\overline{l{:}A'_L}, \overline{m{:}B_L}\}} \leq_\& \qquad \frac{\forall i \in \overline{l} \quad A_{i_L} \leq A'_{i_L}}{\&\{\overline{l{:}A_L}, \overline{m{:}B_L}\} \leq \&\{\overline{l{:}A'_L}\}} \leq_\&$$

One key observation is that subtyping of session types are covariant in their continuations.

Example Since priority queues can be used to sort a list of n objects by repeatedly inserting all n objects first and then popping as in heapsort, we can use subtyping to make this behavior explicit. We begin by sketching the session type of a (priority) queue of some linear type A_L:

$$\begin{aligned} \text{queue} = \&\{&\mathit{insert}: A_L \multimap \text{queue}, \\ &\mathit{pop}: \oplus\{\mathit{some}: A_L \otimes \text{queue}, \mathit{none}: 1\}\} \end{aligned}$$

The session type begins with an external choice, forcing the client to choose between an insert, which requires the client to further send a channel of type A_L and continue as a queue, and a pop, which allows the provider to send a channel of type A_L and continue as a queue or terminate if the queue is already empty. At the current stage, clients using these queues can arbitrarily choose a sequence of insert/pop commands, such as inserting twice, popping once, inserting three times, and so on. To enforce the aforementioned behavior of inserting everything first and then only popping, we define two *phases* of communication for the client. In the first phase, it can insert as many times as the client wants until the client chooses to pop. Upon the first pop, the client transitions to the second phase where it can only pop and never insert.

$$\begin{aligned} \text{queue\_one} = \&\{&\mathit{insert}: A_L \multimap \text{queue\_one}, \\ &\mathit{pop}: \oplus\{\mathit{some}: A_L \otimes \text{queue\_two}, \mathit{none}: 1\}\} \\ \text{queue\_two} = \&\{&\mathit{pop}: \oplus\{\mathit{some}: A_L \otimes \text{queue\_two}, \mathit{none}: 1\}\} \end{aligned}$$

Where the client begins in the first phase, or queue_one, and we can indeed verify that queue $\leq$ queue_one. A client with this view can begin by inserting as many times as it wants. However, once it pops once, it must commit to popping because the session type switches to queue_two which only allows pop, allowing us to encode the two phases of communication.

However, it is important to note that in this particular situation, the provider can also adopt queue_one as its view on the session type. This does not however make subtyping on purely linear types useless; a provider implementing a generic priority queue can be reused to implement a heapsort using subtyping, which a system without subtyping cannot do without re-implementing the provider with queue_one. Once shared channels come to play however, the idea that the provider simply adopts the client's view cannot apply since there can be multiple clients each with different views of the type of a shared channel.

3.2 Subtyping of Modal Shifts

We will begin this section with a motivating example and develop the theory. Consider a shared process that implements a binary choice voting machine. A potential session type to represent this communication begins with an upshift followed by an external choice with three options:

$$\begin{aligned} \text{voting} = {\uparrow}_L^S \& \{ \mathit{vote}1 :& {\downarrow}_L^S \text{voting}, \\ \mathit{vote}2 :& {\downarrow}_L^S \text{voting}, \\ \mathit{result} :& \text{nat} \otimes \text{nat} \otimes {\downarrow}_L^S \text{voting} \} \end{aligned}$$

where the first two labels encode a particular vote and the final *result* label makes the provider respond with the voting counts of the two voting choices.

We can immediately observe that if a process was given a channel of this type, it can vote as many times as it likes, which is not the intention of the voting machine – we would like to enforce that clients only vote once (but can view the result as many times as they like). Thus, from the client's perspective, there are two *phases* to the protocol. The first phase allows the client to see results as many times as it wants and make a vote. Once the client makes a vote, it transitions to the second phase where it can no longer vote but can still view results as many times as it likes. There are two obvious subtyping relations using the shifts that follow our pattern of subtyping so far that we can try to take advantage of:

$$\frac{A_L \leq B_L}{{\uparrow}_L^S A_L \leq {\uparrow}_L^S B_L} \leq_{{\uparrow}_L^S} \qquad \frac{A_S \leq B_S}{{\downarrow}_L^S A_S \leq {\downarrow}_L^S B_S} \leq_{{\downarrow}_L^S}$$

Using these rules, we first attempt to follow the previous queue example and limit the labels that the client can send. For example, we can construct a candidate supertype of voting, voting_one that aims to capture the "each client can only vote once" constraint:

$$\begin{aligned} \text{voting\_one} = {\uparrow}_L^S \& \{ \mathit{vote}1 :& {\downarrow}_L^S \text{voting\_two}, \\ \mathit{vote}2 :& {\downarrow}_L^S \text{voting\_two}, \\ \mathit{result} :& \text{nat} \otimes \text{nat} \otimes {\downarrow}_L^S \text{voting\_one} \} \\ \text{voting\_two} = {\uparrow}_L^S \& \{ \mathit{result} :& \text{nat} \otimes \text{nat} \otimes {\downarrow}_L^S \text{voting\_two} \} \end{aligned}$$

This type seems plausible, since after committing to a particular vote, the client will release to a type voting_two which only allows viewing of the result. However, there is another critical issue. Since shared channels can be duplicated or aliased via contraction, a malicious client can still vote as many times as it wants by ignoring the released shared channel of type voting_two and instead repeatedly using multiple aliases of the initial channel of type voting_one. In general, the strategy of encoding phases in communication through a bigger shared session type allows malicious clients to interact from any previous phases at will. What we really need is a way to relate a shared type and a linear type since linear channels cannot be duplicated, avoiding this issue altogether.

It turns out that the heart of this work is the discovery of a subtyping relation between shared types and linear types – other differences and extensions that we later introduce are naturally derived from this initial step. We first add two new linear connectives $\uparrow_L^L$ and $\downarrow_L^L$ that are operationally equivalent to $\uparrow_L^S$ and $\downarrow_L^S$, respectively, from a protocol perspective. Thus, the protocol denoted by $\uparrow_L^L A_L$ requires the client to acquire as in the shared case. If the provider happens to implement the same $\uparrow_L^L A_L$, then there is no merit to this connective since a linear channel already enforces exclusive access between the client and provider. The more interesting case is when the provider is actually providing a shared channel, some $\uparrow_L^S A_L$, a client should be able to view that the session type is $\uparrow_L^L A_L$ without any trouble. This idea formalizes the following subtyping relations:

$$\frac{A_L \leq B_L}{\uparrow_L^S A_L \leq \uparrow_L^L B_L} \leq_{\uparrow_L^S \uparrow_L^L} \quad \frac{A_S \leq B_L}{\downarrow_L^S A_S \leq \downarrow_L^L B_L} \leq_{\downarrow_L^S \downarrow_L^L}$$

$$\frac{A_L \leq B_L}{\uparrow_L^L A_L \leq \uparrow_L^L B_L} \leq_{\uparrow_L^L} \quad \frac{A_L \leq B_L}{\downarrow_L^L A_L \leq \downarrow_L^L B_L} \leq_{\downarrow_L^L}$$

Using the new connectives, we can finish our voting machine example; a client must actually view the voting machine as a linear channel!

$$\begin{aligned}
\text{voting\_one} = \uparrow_L^L \& \{ &vote1 : \downarrow_L^L \text{voting\_two}, \\
&vote2 : \downarrow_L^L \text{voting\_two}, \\
&result : \text{nat} \otimes \text{nat} \otimes \downarrow_L^L \text{voting\_one}\} \\
\text{voting\_two} = \uparrow_L^L \& \{ &result : \text{nat} \otimes \text{nat} \otimes \downarrow_L^L \text{voting\_two}\}
\end{aligned}$$

Therefore, the addition of the “trivial” purely linear shift operators $\uparrow_L^L$ and $\downarrow_L^L$ allows us to relate shared session types and linear session types. Although these operators are not useful by themselves because the acquire-release paradigm for purely linear session types provide no practical value, they serve a relevant role in a system with subtyping.

3.3 Safely Synchronizing

With the adoption of manifest sharing, shared session types are required to adhere to an equi-synchronizing constraint as formulated in [BP17] with the intention that we must release a channel to the same state at which it was acquired. For example, a shared session type such as $A_S = \uparrow_L^S \&\{a : \downarrow_L^S A_S, b : \downarrow_L^S B_S\}$ for some unrelated B_S would not be equi-synchronizing since there is a release point B_S that does not return A_S to its initial type before the acquire.

The adoption of subtyping to our system allows us to relax the constraint that the channel must return to its original type; this is equivalent to the formulation of *sub-synchronizing types* [San20; Das+21] (see Section 5) except we take a provider-centric judgment as opposed to the client-centric judgment. Consider some provider that provides a channel x_S of type A_S and n clients of x_S each with their own perception of the type of x: $A_1, A_2, \ldots, A_n$ such that for all $i \leq n$, $A_S \leq A_i$. Then it seems perfectly reasonable to allow future release points of A_S to be a smaller type B_S, since by transitivity, $B_S \leq A_i$ for all $i \leq n$. Therefore, we modify the equi-synchronizing constraint to allow types to be released at a smaller type. More generally, we extend shared types $\hat{A} ::= \bot \mid A_S \mid \top$ with $\bot \leq A_S \leq \top$ for any A_S. Intuitively, $\top$ indicates a channel that has never been acquired (no constraints on a future release), A_S indicates the previous presentation of shared channels, and $\bot$ indicates a channel that will never be available (hence, any client attempting to acquire from this channel will never succeed and be blocked).

With the introduction of subtyping, it is now important to further refine the notion of equi-synchronizing types to take into account the possibility that a provider and client of a channel have different views of its session type, where the provider's view is a subtype of the client's view. Recall the example $A_S = \uparrow_L^S \&\{a : \downarrow_L^S A_S, b : \downarrow_L^S B_S\}$ where B_S is unrelated to A_S which we claimed is not equi-synchronizing. However, suppose a provider offering a channel $x_S{:}A_S$ only has one client which views x_S as A'_S where $A'_S = \uparrow_L^S \&\{a : \downarrow_L^S A_S\}$ (we can verify that $A_S \leq A'_S$). In this case, we are aware that the label b will never be sent by the client, and therefore, the provider will never receive a label b. To take advantage of these scenarios where the provider and client do not view a channel as the same type, the synchronizing constraint is no longer a restriction on a session type but instead a restriction on a pair of session types C and D such that $C \leq D$ where we interpret a provider to provide a channel of type C and a client to locally view the channel as type D.

We are now ready to present the *safely synchronizing* judgment, interpreted coinductively, which is of form $\vdash (A, B, \hat{D})$ dfsync for some A and B such that $A \leq B$, which asserts that a provider providing a channel of type A and a client using that channel with type B is safely synchronizing with respect to some constraint $\hat{D}$:

$$\frac{}{\vdash (1, 1, \hat{D}) \text{ dfsync}} D1 \quad \frac{\vdash (B_L, B'_L, \hat{D}) \text{ dfsync}}{\vdash (A_L \otimes B_L, A'_L \otimes B'_L, \hat{D}) \text{ dfsync}} D\otimes \quad \frac{\vdash (B_L, B'_L, \hat{D}) \text{ dfsync}}{\vdash (A_L \multimap B_L, A'_L \multimap B'_L, \hat{D}) \text{ dfsync}} D\multimap$$

$$\frac{\forall i \in \overline{l} \quad \vdash (A_{i_L}, A'_{i_L}, \hat{D}) \text{ dfsync}}{\vdash (\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}A'_L}, \overline{m{:}B_L}\}, \hat{D}) \text{ dfsync}} D\oplus \quad \frac{\forall i \in \overline{l} \quad \vdash (A_{i_L}, A'_{i_L}, \hat{D}) \text{ dfsync}}{\vdash (\&\{\overline{l{:}A_L}, \overline{m{:}B_L}\}, \&\{\overline{l{:}A'_L}\}, \hat{D}) \text{ dfsync}} D\&$$

$$\frac{\vdash (A_L, A'_L, \hat{D}) \text{ dfsync}}{\vdash (\uparrow_L^L A_L, \uparrow_L^L A'_L, \hat{D}) \text{ dfsync}} D\uparrow_L^L \quad \frac{\vdash (A_L, A'_L, \hat{D}) \text{ dfsync}}{\vdash (\downarrow_L^L A_L, \downarrow_L^L A'_L, \hat{D}) \text{ dfsync}} D\downarrow_L^L$$

$$\frac{\vdash (A_L, A'_L, \uparrow_L^S A_L) \text{ dfsync}}{\vdash (\uparrow_L^S A_L, \uparrow_L^S A'_L, \top) \text{ dfsync}} D\uparrow_L^S \quad \frac{\vdash (A_S, A'_S, \top) \text{ dfsync} \quad \downarrow_L^S A_S \leq \hat{D}}{\vdash (\downarrow_L^S A_S, \downarrow_L^S A'_S, \hat{D}) \text{ dfsync}} D\downarrow_L^S$$

$$\frac{\vdash (A_L, A'_L, \uparrow_L^S A_L) \text{ dfsync}}{\vdash (\uparrow_L^S A_L, \uparrow_L^L A'_L, \top) \text{ dfsync}} D\uparrow_L^S\uparrow_L^L \quad \frac{\vdash (A_S, A'_L, \top) \text{ dfsync} \quad \downarrow_L^S A_S \leq \hat{D}}{\vdash (\downarrow_L^S A_S, \downarrow_L^L A'_L, \hat{D}) \text{ dfsync}} D\downarrow_L^S\downarrow_L^L$$

where the general intuition is that when encountering internal and external choices, we take the intersection of the choices – after all, the branches that the two types do not have in common will never occur during communication.

3.4 System

Finally, we can present our system $SILL_{S\leq}$. For economic reasons, we will only present some interesting rules – the full system along with the technical details of the relevant meta theory are presented in the Appendix: Appendix A for the subtyping rules including extended types, Appendix C for the safely synchronizing judgments, Appendix D for the type judgments, Appendix E for the operational semantics, and Appendix B for the configuration judgments.

3.4.1 Configuration

A configuration consists of a list of shared processes Λ and a list of linear processes Θ. The order of shared processes have no structure, but the order of linear processes can be seen to form a tree structure; a linear process can use channels offered by processes to its right, and due to linearity, if it is using a channel, it must be the unique process doing so.

$$\begin{aligned}\Omega &::= \Lambda; \Theta \\ \Lambda &::= \cdot \mid \Lambda_1, \Lambda_2 \mid \mathrm{proc}(a_S, P) \mid \mathrm{unavail}(a_S) \\ \Theta &::= \cdot \mid \mathrm{proc}(a_L, P), \Theta' \mid \mathrm{connect}(a_L, b_S), \Theta'\end{aligned}$$

The connect term is a new addition compared to $SILL_S$ [BP17] and is used as a primitive to make subtyping between linear channels and shared channels explicit.

We also introduce the meta variable Ψ as a shorthand to represent a linear process:

$$\Psi_a ::= \mathrm{proc}(a_L, P) \mid \mathrm{connect}(a_L, b_S)$$

Ψ is parametrized by a subscript, such as a in Ψ_a, to make explicit the offering channel name a_L. Uses of Ψ without a subscript generally means the name of the offering channel is not relevant in the context and therefore can be considered a shorthand for "Ψ_a for some a".

A (well-formed[6]) configuration $\Omega ::= \Lambda; \Theta$ is typed by its shared and linear fragments.

$$\frac{\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta :: (\Delta)}{\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)}\ \Omega$$

3.4.2 Shared Fragment Λ

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Lambda 1 \qquad \frac{\Gamma \models \Lambda_1 :: (\Gamma_1) \quad \Gamma \models \Lambda_2 :: (\Gamma_2)}{\Gamma \models \Lambda_1, \Lambda_2 :: (\Gamma_1, \Gamma_2)}\ \Lambda 2$$

$$\frac{\vdash (A'_S, A_S, \top)\ \mathrm{dfsync} \quad \Gamma \vdash P :: (a_S{:}A'_S)}{\Gamma \models \mathrm{proc}(a_S, P) :: (a_S{:}A_S)}\ \Lambda 3 \qquad \frac{}{\Gamma \models \mathrm{unavail}(a_S) :: (a_S{:}\hat{A})}\ \Lambda 4$$

[6]see Appendix B

3.4.3 Linear Fragment Θ

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Theta1 \qquad \frac{b_S{:}\hat{B} \in \Gamma \quad b_S \leq A_L \quad \Gamma \models \Theta' :: (\Delta')}{\Gamma \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, \Delta')}\ \Theta2$$

$$\frac{a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync} \quad \Gamma; \Delta_a \vdash P :: (a_L{:}A'_L) \quad \Gamma \models \Theta' :: (\Delta_a, \Delta')}{\Gamma \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')}\ \Theta3$$

Some notable rules are $\Lambda3$ and $\Theta3$, where in both cases, the process typing judgment in the premise offers a channel type A' that is different from the one offered in the configuration A. The subtyping relation of $A' \leq A$ is implicit through the safely synchronizing constraint $\vdash (A', A, \hat{A})$ dfsync. Another notable rule is $\Theta2$, which formalizes the role of the connect term as a “middleman” between a linear channel and a shared channel.

3.4.4 Type judgments

Many of the rules are very similar to the ones defined in Section 2.2.1. We present some cases that differ due to our subtyping system. First, we relax the two identity rules to allow forwarding with a smaller type. We also allow a linear process to forward using a shared channel if the appropriate subtyping relation holds.

$$\frac{B_L \leq A_L}{\Gamma; y_L{:}B_L \vdash \text{fwd}\ x_L\ y_L :: (x_L{:}A_L)}\ ID_L \qquad \frac{\hat{B} \leq A_S}{\Gamma, y_S{:}\hat{B} \vdash \text{fwd}\ x_S\ y_S :: (x_S{:}A_S)}\ ID_S \qquad \frac{\hat{B} \leq A_L}{\Gamma, y_S{:}\hat{B}; \cdot \vdash \text{fwd}\ x_L\ y_S :: (x_L{:}A_L)}\ ID_{LS}$$

Shared channels in the context may now have an extended type, needed to express the subtle constraints imposed by safe synchronization. For the most part, there is not much loss in the high level details if one were to interpret these constraints as just shared session types A_S.

A linear to linear spawn (a named cut) now divides the channel substitutions to three parts: linear to linear substitutions, shared to linear substitutions, and shared to shared substitutions. The shared to linear substitution in particular occurs when a process definition expects a linear channel (or some type $\uparrow_L^L \ldots$) and is instead given a smaller shared channel, and is in fact the key to the expressiveness of our system.

$$\frac{\begin{matrix}\overline{v_S{:}\hat{D}{\in}\Gamma} \\ \overline{w_S{:}\hat{E}{\in}\Gamma}\end{matrix} \quad \begin{matrix}\overline{B_L}{\leq}\overline{B'_L} \\ \overline{\hat{D}}{\leq}\overline{D'_L} \\ \overline{\hat{E}}{\leq}\overline{E'_S}\end{matrix} \quad \big(x'_L{:}(A_L \leq A'_L) \leftarrow X_L \leftarrow \overline{y'_L{:}B'_L}, \overline{v'_L{:}D'_L}, \overline{w'_S{:}E'_S} = P\big) \in \Sigma \quad \Gamma; \Delta, x_L{:}A'_L \vdash Q :: (z_L{:}C_L)}{\Gamma; \Delta, \overline{y_L{:}B_L} \vdash x_L \leftarrow X_L \leftarrow \overline{y_L}, \overline{v_S}, \overline{w_S}; Q :: (z_L{:}C_L)}\ SP_{LL}$$

Note the offering channel of the process X_L in the signature being defined by a pair of types $x'_L{:}(A_L \leq A'_L)$. This makes explicit the perspective that from the provider's perspective (X_L), the channel is viewed as type A_L and from the client's perspective, the channel is viewed as type A'_L where $A_L \leq A'_L$.

We also allow sending of channels with smaller types:

$$\frac{\Gamma; \Delta, x_L{:}B_L, y_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}A_L \otimes B_L \vdash y_L \leftarrow \text{recv } x_L; P :: (z_L{:}C_L)} \otimes L \quad \frac{A'_L \leq A_L \quad \Gamma; \Delta \vdash P :: (x_L{:}B_L)}{\Gamma; \Delta, y_L{:}A'_L \vdash \text{send } x_L\ y_L; P :: (x_L{:}A_L \otimes B_L)} \otimes R$$

Similar to the rules presented before, there is also a version where we can send a shared channel as long as its type is smaller than the expected (linear) type.

Remark Since typing judgments are purely local to a process, most of the subtyping occurs as special cases of certain connectives; the configuration keeps track of persisting any subtyping that arises.

3.4.5 Dynamics

We adopt a synchronous communication pattern in our system. As noted in [PG15], asynchronous communication can be easily adopted in this system aside from the acquire step. Following Cervesato and Scedrov [CS09] and prior work [BP17], we formalize the operational semantics as *multiset rewriting rules*, which is of the form

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

and denote a computational step from the left terms to the right terms. In our paper, the terms S and T correspond to terms in the configuration, namely $\text{proc}(a_S, P), \text{unavail}(a_S), \text{proc}(a_L, P)$, and $\text{connect}(a_L, b_S)$. We present some interesting operational semantics rules in this section; the complete listing of the rules are available in Appendix E.

If a linear process is waiting on some channel b_L to close and another linear process offering b_L is attempting to close the channel, then the process offering b_L can safely terminate:

$$\text{proc}(a_L, \text{wait } b_L; P), \text{proc}(b_L, \text{close } b_L) \to \text{proc}(a_L, P) \qquad \text{(D-1)}$$

Similarly, if a linear process is waiting to receive a channel from b_L and a process offering b_L is sending a channel through b_L, the channel is sent from one process to another:

$$\begin{aligned}
&\text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_L; Q), \Psi_c && (\text{D-}\otimes)\\
&\quad\to \text{proc}(a_L, [c_L/y_L]P), \text{proc}(b_L, Q), \Psi_c\\
&\text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_S; Q) && (\text{D-}\otimes 2)\\
&\to \text{proc}(a_L, [d_L/y_L]P), \text{proc}(b_L, Q), \text{connect}(d_L, c_S), \text{unavail}(d_S) \quad (d \text{ fresh})
\end{aligned}$$

The second rule handles the case when a shared channel is sent in place of a linear channel – in that case, a connect term appears and connects a fresh channel d_L to the sent channel c_S. In general, connect terms appear whenever shared channels are interpreted as linear channels due to subtyping.

If a linear process is trying to acquire a channel b_S while a shared process is accepting, then the shared process transitions to its linear form while an unavail(b_S) is left behind to signify that the b_S cannot be acquired by another client in the meantime. The second rule consumes the connect – if similar to the first case, a client is trying to (linearly) acquire a channel b_L that is being connected to some c_S by a connect(b_L, c_S) term, similar results to the first rule occur.

$$\begin{aligned}
&\text{proc}(a_L, x_L \leftarrow \text{acq}_S\ b_S; P), \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q) && (\text{D-}\uparrow_L^S)\\
&\quad\to \text{proc}(a_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L]Q), \text{unavail}(b_S)\\
&\text{proc}(a_L, x_L \leftarrow \text{acq}_L\ b_L; P), \text{connect}(b_L, c_S), \text{proc}(c_S, x_L \leftarrow \text{acc}_S\ c_S; Q) && (\text{D-}\uparrow_L^S 2)\\
&\quad\to \text{proc}(a_L, [c_L/x_L]P), \text{proc}(c_L, [c_L/x_L]Q), \text{unavail}(c_S)
\end{aligned}$$

3.4.6 Theorems

We begin with the preservation theorem, which guarantees session fidelity: a process communicating along a channel will follow the protocol as denoted by the channel.

Theorem 1. *If* $\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)$ *and* $\Lambda; \Theta \to \Lambda'; \Theta'$, *then* $\Gamma' \models \Lambda'; \Theta' :: (\Gamma'; \Delta)$ *for some* Λ', Θ', *and* Γ' *such that* $\Gamma' \preceq \Gamma$.

Proof. See Appendix F. □

The $\Gamma' \preceq \Gamma$ captures the idea that the configuration can gain additional shared processes and that the types of shared channels can become smaller (see Appendix A). For example, if a process spawns an additional shared process, then the configuration will gain an additional channel in Γ and if a shared channel is released to a smaller type, the type of the shared channel in Γ can become smaller. Note that although it is indeed true that linear processes can be spawned, it will never appear in Δ since the linear channel that the newly spawned process offers must be consumed by another process for it to be spawned.

To state the progress theorem, we need to introduce two additional concepts: *poised processes* [PG15] and *blocked processes* [BP17].

Definition 1. A shared and linear process term $\text{proc}(a, P)$ is *poised* if P is currently communicating along its providing channel a. Poised process terms in $SILL_{S\leq}$ are shown in the table below.

Receiving	Sending
	$\text{proc}(a_L, \text{close } a_L)$
$\text{proc}(a_L, x_L \leftarrow \text{recv } a_L; P)$	$\text{proc}(a_L, \text{send } a_L\ c_L; P)$
$\text{proc}(a_L, x_S \leftarrow \text{recv } a_L; P)$	$\text{proc}(a_L, \text{send } a_L\ c_S; P)$
$\text{proc}(a_L, \text{case } a_L \text{ of } \{\overline{l \Rightarrow P}\})$	$\text{proc}(a_L, a.i; P)$
$\text{proc}(a_L, x_L \leftarrow \text{acc}_L\ a_L; P)$	$\text{proc}(a_L, x_L \leftarrow \text{det}_L\ a_L; P)$
$\text{proc}(a_S, x_L \leftarrow \text{acc}_S\ a_S; P)$	$\text{proc}(a_S, x_L \leftarrow \text{det}_S\ a_S; P)$

In particular, we say that a configuration is poised if all of its $\text{proc}(-,-)$ members are poised.

Definition 2. A linear process is *blocked* along a_S if it is attempting to acquire access to a_S which is currently provided by an $\text{unavail}(a_S)$ term. Formally, $\text{proc}(c_L, -)$ is blocked along a_S if the configuration is of form:

1. $\text{proc}(c_L, x_L \leftarrow \text{acq}_S\ a_S), \text{unavail}(a_S)$ or
2. $\text{proc}(c_L, x_L \leftarrow \text{acq}_L\ b_L), \text{connect}(b_L, a_S), \text{unavail}(a_S)$

The first form captures the case when $\text{proc}(c_L, -)$ is directly trying to acquire a_S, while the second form captures the case when $\text{proc}(c_L, -)$ is indirectly trying to acquire a_S through a connect term.

Now we are ready to state the progress theorem, which states that the only way the configuration can be stuck is through a blocked process (for example, a deadlock).

Theorem 2. *If* $\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)$ *then either:*

1. $\Lambda \rightarrow \Lambda'$ *for some* Λ' *or*

2. Λ *poised and one of:*

   (a) $\Lambda; \Theta \rightarrow \Lambda'; \Theta'$ *for some* Λ' *and* Θ' *or*

   (b) Θ *poised or*

   (c) some linear process $\Psi \in \Theta$ *is blocked*

Proof. See Appendix G □

4 Examples

4.1 Deadlock Detection

We base this example from a previous work [San19; WPP17] of a centralized form of Mitchell and Merritt's distributed deadlock detection algorithm [MM84]. The algorithm assumes a distributed system with shared resources and linear nodes, where the intended behavior is that the linear nodes, encoded as linear processes, acquire particular resources, encoded as shared processes, perform appropriate computations, and then release unneeded resources as in typical distributed systems. Both nodes and resources are identified by a unique identification of type pid (process id) and rid (resource id) respectively, which as in previous examples, we take as primitives. In this system, a deadlock in the usual sense is detected when there is a cycle in the dependency graph generated by the algorithm. The centralized deadlock detection algorithm consists of a shared process that acts as an overseer that all nodes report to.

The type of this global deadlock detection overseer is given as

$$\begin{aligned} \mathrm{dd} = {\uparrow}^S_L \& \{ & \mathit{tryacq} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd}, \\ & \mathit{didacq} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd}, \\ & \mathit{willrel} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd} \} \end{aligned}$$

where the intention is that clients are expected to inform the overseer before attempting to acquire a resource (*tryacq*), after successfully acquiring a resource (*didacq*), and before releasing a resource (*willrel*).

As discussed in a previous work [San19], there are two phases of the protocol across successive acquire-release cycles. Using subtyping, we can represent this constraint statically:

$$\begin{aligned} \mathrm{dd\_one} = {\uparrow}^L_L \& \{ & \mathit{tryacq} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_two}, \\ & \mathit{willrel} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_one} \} \\ \mathrm{dd\_two} = {\uparrow}^L_L \& \{ & \mathit{didacq} : \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_one} \} \end{aligned}$$

where $\mathrm{dd} \leq \mathrm{dd\_one}$. This session type enforces that the message following *tryacq* must be *didacq* and that *didacq* cannot be sent before a *tryacq*. It is important to note that we are not enforcing other desirable constraints such as whether the resource id sent by the client matches in a sequence of $\mathit{tryacq} \to \mathit{didacq}$ (it is nonsensical for a client to attempt to acquire resource r and after claim that it successfully acquired a different resource r'). We believe that those additional constraints can be naturally expressed by extending refinement types [DP20] to be compatible with this system.

A linear node is a process that uses a channel of type dd_one; since we allow subtyping across modalities, we can spawn such a node by passing a reference to the global overseer offering a shared channel of type dd, which the node can safely view to be dd_one since dd $\leq$ dd_one.

Since the overseer is locally seen as a linear channel for the nodes, the linearity condition along with dd_one being infinite means that the nodes can never terminate, which is a difference from standard implementations without subtyping where nodes can terminate freely since in that setting, the clients hold shared references to the overseer. To resolve this, we can attempt to expand the protocol to introduce explicit join and exit messages:

$$\begin{aligned}
\mathrm{dd} = {\uparrow}^S_L \& \{ \mathit{tryacq} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd}, \\
\mathit{didacq} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd}, \\
\mathit{willrel} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^S_L \mathrm{dd}, \\
\mathit{join} &: \mathrm{pid} \multimap {\downarrow}^S_L \mathrm{dd}, \\
\mathit{exit} &: \mathrm{pid} \multimap {\downarrow}^S_L \mathrm{dd} \}
\end{aligned}$$

From the client's perspective, we have:

$$\begin{aligned}
\mathrm{dd\_entry} = {\uparrow}^S_L \& \{ \mathit{join} &: \mathrm{pid} \multimap {\downarrow}^L_L \mathrm{dd\_one} \} \\
\mathrm{dd\_one} = {\uparrow}^L_L \& \{ \mathit{tryacq} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_two}, \\
\mathit{willrel} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_one}, \\
\mathit{exit} &: \mathrm{pid} \multimap {\downarrow}^S_L \mathrm{dd\_entry} \} \\
\mathrm{dd\_two} = {\uparrow}^L_L \& \{ \mathit{didacq} &: \mathrm{pid} \multimap \mathrm{rid} \multimap {\downarrow}^L_L \mathrm{dd\_one} \}
\end{aligned}$$

where dd $\leq$ dd_entry. Nodes now begin with a channel of type dd_entry, a shared type. The key change is that in dd_one, clients can now send *exit*, which brings back a shared type, allowing nodes to terminate.

Unfortunately, although forcing the client to view the channel's type as dd_entry seems to give a complete solution to the exiting dilemma, this creates a vulnerability in the system. For example, malicious clients can join twice since the initial dd_entry is a shared channel and therefore can be duplicated. Although there is still value in this extension if we assume innocent clients, we believe that a full general solution requires incorporating affine modality to our system which we leave for future work (see Section 6.1).

4.2 Auction

Next we consider a variation of the auction protocol described by Das et al. [Das+21], where an auction consists of a shared provider acting as the auctioneer and its multiple clients acting as bidders. The auction transitions between the betting phase where clients are allowed to place bets and the collecting phase where a winner is given the auctioned item while all the losers are refunded their respective bids. Taking the linear types id, money, and item as primitives, we have:

$$\begin{aligned}
\text{auction} = {\uparrow}^S_L \& \{ &\mathit{bid} : \oplus \{ \mathit{ok} : \text{id} \multimap \text{money} \multimap {\downarrow}^S_L \text{auction}, \\
&\qquad\qquad \mathit{collecting} : {\downarrow}^S_L \text{auction} \}, \\
&\mathit{collect} : \oplus \{ \mathit{prize} : \text{id} \multimap \text{item} \otimes {\downarrow}^S_L \text{auction}, \\
&\qquad\qquad \mathit{refund} : \text{id} \multimap \text{money} \otimes {\downarrow}^S_L \text{auction}, \\
&\qquad\qquad \mathit{bidding} : {\downarrow}^S_L \text{auction} \} \}
\end{aligned}$$

From the perspective of the client, it must first acquire, then either choose between *bid* or *collect*. If it chooses *bid*, the auctioneer will either respond with an *ok*, indicating that the auction is currently in the bidding phase, allowing the client to send its id, money, and then release back to the auction type, or a *collecting*, indicating that the auction is in the collect phase, making the client release back to the auction type. If the client chooses *collect*, then the auctioneer will respond with one of $\mathit{prize}, \mathit{refund}$, or *bidding*. The *prize* branch covers the case where the client won the previous bid, the refund branch covers the case where the client lost the bid, and the *bidding* branch informs the client that the auction is currently in the bidding phase.

Like in previous examples, the auction protocol has distinct phases of communication, in particular the bidding phase and the collecting phase. Following previous examples, we can construct a supertype of the auction type, auction_bidding to make these phases more explicit for the client:

$$\begin{aligned}
\text{auction\_bidding} &= {\uparrow}^L_L \& \{ \mathit{bid} : \oplus \{ \mathit{ok} : \text{id} \multimap \text{money} \multimap {\downarrow}^L_L \text{auction\_bidding}, \\
&\qquad\qquad\qquad\quad \mathit{collecting} : {\downarrow}^L_L \text{auction\_collecting} \} \} \\
\text{auction\_collecting} &= {\uparrow}^L_L \& \{ \mathit{collect} : \oplus \{ \mathit{prize} : \text{id} \multimap \text{item} \otimes {\downarrow}^L_L \text{auction\_bidding}, \\
&\qquad\qquad\qquad\quad \mathit{refund} : \text{id} \multimap \text{money} \otimes {\downarrow}^L_L \text{auction\_bidding}, \\
&\qquad\qquad\qquad\quad \mathit{bidding} : {\downarrow}^L_L \text{auction\_bidding} \} \}
\end{aligned}$$

where auction $\leq$ auction_bidding.

Aside from making the phases in the protocol more explicit, the adoption of auction_bidding guarantees that clients take into account the *collecting* and *bidding* responses of the provider and allows the programming pattern of the provider "choosing" the subsequent phase to be manifest in the type itself. Furthermore, this particular presentation of the auction protocol paves way for future extensions involving refinement types [DP20] that can allow static verification of more intricate properties of the protocol for example a property such as if a participant puts in n units of money in the bidding phase and does not win, it will be refunded exactly n units of money. Although we face a similar exit problem as in the previous example, similar strategies can be taken to allow clients to terminate.

5 Related Work

Our work serves as an extension to $SILL_S$ defined in [BP17] by introducing a notion of subtyping to the system which allows us to statically relax the equi-synchronizing constraint present in $SILL_S$. Early glimpses of subtyping can be seen in the previous system with the introduction of $\bot$ and $\top$ as the minimal and maximal constraints, which happened to coincide with the minimal and maximal shared types $\bot$ and $\top$ in our much more expansive subtyping relation.

More generally, our work is based on many past works on session types, with a proposal for a type system for the π-calculus [Hon93; HVK98], and amongst many works that further develop this theory, a subtyping extension to the system [GH05]. Variations of the earlier type system was later given a correspondence with intuitionistic linear logic [CP10; Ton15] and classical linear logic [Wad12]. The sharing semantics that we investigate in this paper was introduced in [BP17] by decomposing the exponential modality, which was previously known to provide a copying semantics through association with the exponential operator in linear logic. The type system arising from the correspondence with intuitionistic linear logic was later given a subtyping extension [AP16], which covered all linear connectives along with union and intersect types (still linear) that we omitted in our work. One stylistic difference between the two is that our subtyping is present directly in the type system, whereas the subtyping in [AP16] is implicitly present through a subsumption rule (and corresponding meta theory to prove its safety).

Sub-synchronizing types [San20; Das+21] is another relaxation of the equi-synchronizing constraint that allows types to be released at a smaller type as initiated by the client. This is captured precisely by the subtyping aspect of our safely synchronizing constraint where the main difference between the two approaches is that our approach incorporates full subtyping, which fixes the provider's type. We therefore take a provider-centric view of the synchronization constraint, which we point is equivalent to the client-centric approach that sub-synchronizing types take had we removed full subtyping in our system since both providers and clients see the equivalent session types. In particular, if A_S is sub-synchronizing, then $\vdash (A_S, A_S, \top)$ dfsync in our system. A drawback to sub-synchronizing types is that shared session types can only get smaller, meaning cyclic phases cannot be expressed. In our system, we avoid this issue by fixing the provider type and only maintaining the invariant that the client's view of the type is bigger, we allow the client to release at not only smaller types but also bigger types and even unrelated types.

There have also been many recent developments in subtyping in the context of multiparty session types [CDCY14; Che+17; Ghi+19; Ghi+20], which are a different class of type systems that describe protocols between an arbitrary number of participants from a neutral global point of view. Understanding the relation of our work to these systems is an interesting item for future work.

6 Discussion and Future Work

We have demonstrated a subtyping extension to a subset of $SILL_S$ and showed many examples highlighting the expressiveness that this new system provides. Because we worked on a relatively minimalistic system to isolate and demonstrate the principal ideas behind subtyping, there is more work to be done to adopt the notion of subtyping that we proposed to some of the recent discoveries and proposals with session typed process calculi. Although we believe that our notion of subtyping is consistent with other extensions (to linear logic based typed process calculi), a verification that our proposed form of subtyping is indeed compatible with different extensions such as asynchronous communication [PG15], refinement types [DP20], etc. would set this subtyping relation onto a firmer ground.

One of the biggest technical contributions that we have made in this work is the careful relaxation of the previously formulated equi-synchronizing constraint, now formulated as the safely synchronizing constraint, which we find is a key step in making the sharing semantics of shared channels practical for programming languages that support message passing concurrency such as Rust [KN19]. It is not clear whether more relaxations can be made at static level, but a more focused investigation on the topic may reveal more natural extensions. Another technical discovery we have made is that the extended shared session types $\hat{A}$ form a lattice. We have found use of the meet operation in our proofs due to the need to find lower bounds of two shared session types when working with the preservation theorem, but we have not yet found a use for the join operation nor a satisfying interpretation for it.

We have demonstrated the need of the trivial shift $\uparrow_L^L$ and $\downarrow_L^L$ to reason about subtyping of modal shifts, which we find is the most insightful discovery in this paper from a foundational perspective. In particular, we have not yet considered in detail the meaning of subtyping in other disciplines that are identified with session types, such as linear logic and modalities from a proof theory perspective and symmetric monoidal closed categories and adjoints from a category theory perspective. In particular, we believe that we can immediately extend our subtyping discipline to adjoint logic [PP19]; we expand on this in the following section.

6.1 Affine Mode

In many of our examples, shared to linear subtyping made it such that clients can never terminate, requiring us to add explicit exit branches in the external choice. We believe we can include an affine mode F that lies between the shared mode S and the linear mode L which rejects contraction but permits weakening. We can stratify the affine mode and add appropriate shifts $\uparrow_L^F$ and $\downarrow_L^F$; we omit S to F, F to S, and F to F shifts for this presentation for economic reasons, with the following subtyping relations:

$$\frac{A_L \leq B_L}{\uparrow_L^S A_L \leq \uparrow_L^F B_L} \leq_{\uparrow_L^S \uparrow_L^F} \quad \frac{A_L \leq B_L}{\uparrow_L^F A_L \leq \uparrow_L^L B_L} \leq_{\uparrow_L^F \uparrow_L^L} \quad \frac{A_S \leq B_F}{\downarrow_L^S A_S \leq \downarrow_L^F B_F} \leq_{\downarrow_L^S \downarrow_L^F} \quad \frac{A_F \leq B_L}{\downarrow_L^F A_F \leq \downarrow_L^L B_L} \leq_{\downarrow_L^F \downarrow_L^L}$$

We can follow the pattern to define the subtyping relation for the remaining shifts $\uparrow_F^S, \uparrow_F^F, \downarrow_F^S$, and $\downarrow_F^F$, although they are not required for this particular example.

We now use these new shifts to fix the termination issue with the voting example in Section 3.2. The provider's view is unchanged:

$$\begin{aligned} \text{voting} = \uparrow_L^S \&\{ &\mathit{vote}1 : \downarrow_L^S \text{voting}, \\ &\mathit{vote}2 : \downarrow_L^S \text{voting}, \\ &\mathit{result} : \text{nat} \otimes \text{nat} \otimes \downarrow_L^S \text{voting}\} \end{aligned}$$

However, the client's view now uses affine modality:

$$\begin{aligned} \text{voting\_one} = \uparrow_L^F \&\{ &\mathit{vote}1 : \downarrow_L^F \text{voting\_two}, \\ &\mathit{vote}2 : \downarrow_L^F \text{voting\_two}, \\ &\mathit{result} : \text{nat} \otimes \text{nat} \otimes \downarrow_L^F \text{voting\_one}\} \\ \text{voting\_two} = \uparrow_L^F \&\{ &\mathit{result} : \text{nat} \otimes \text{nat} \otimes \downarrow_L^F \text{voting\_two}\} \end{aligned}$$

where the client views the type of the channel as voting_one, which is affine. Since contraction is rejected in the affine mode, we do not have to worry about the issue with shared modes where clients can ignore the freshly released channels and repeatedly use the initial channel. However, since weakening is allowed in the affine mode, (linear) clients can freely terminate while holding a reference to the affine channel.

More generally, we can work in adjoint logic which allows us to reason about arbitrary modalities following a partial order that is monotonic in the set of substructural rules that it allows. For example, in the affine extension we have described, the partial order of modalities go $L \leq F \leq S$, and we can indeed verify that the set of substructural rules for each modality respects the partial order: $\{E\} \subseteq \{E, W\} \subseteq \{E, W, C\}$ where E stands for exchange, W for weakening, and C for contraction.

References

[AP16] Coşku Acay and Frank Pfenning. "Intersections and Unions of Session Types". In: *8th Workshop on Intersection Types and Related Systems (ITRS'16)*. Ed. by N. Kobayashi. Porto, Portugal: EPTCS 242, June 2016, pp. 4–19.

[BP17] Stephanie Balzer and Frank Pfenning. "Manifest Sharing with Session Types". In: *International Conference on Functional Programming (ICFP)*. Extended version available as Technical Report CMU-CS-17-106R, June 2017. ACM, Sept. 2017, 37:1–37:29.

[CDCY14] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. "On the Preciseness of Subtyping in Session Types". In: *Proceedings of the Conference on Principles and Practice of Declarative Programming (PPDP'14)*. Canterbury, UK: ACM, Sept. 2014.

[Che+17] Tzu-chun Chen et al. "On the Preciseness of Subtyping in Session Types". In: *Logical Methods in Computer Science* Volume 13, Issue 2 (June 2017). DOI: `10.23638/LMCS-13(2:12)2017`. URL: `https://lmcs.episciences.org/3752`.

[CHP99] Karl Crary, Robert Harper, and Sidd Puri. "What is a Recursive Module?" In: *In SIGPLAN Conference on Programming Language Design and Implementation*. ACM Press, 1999, pp. 50–63.

[CP10] Luís Caires and Frank Pfenning. "Session Types as Intuitionistic Linear Propositions". In: *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*. Paris, France: Springer LNCS 6269, Aug. 2010, pp. 222–236.

[CS09] Iliano Cervesato and Andre Scedrov. "Relating State-Based and Process-Based Concurrency through Linear Logic". In: *Information and Computation* 207.10 (Oct. 2009), pp. 1044–1077.

[Das+21] Ankush Das et al. "Resource-Aware Session Types for Digital Contracts". In: *21st Symposium on Computer Security Foundation (CSF-21)*. IEEE. 2021, forthcoming.

[DP20] Ankush Das and Frank Pfenning. "Session Types with Arithmetic Refinements". In: *31st International Conference on Concurrency Theory (CONCUR 2020)*. Ed. by I. Konnov and L. Kovács. Vienna, Austria: LIPIcs 171, Sept. 2020, 13:1–13:18.

[GH05] Simon J. Gay and Malcolm Hole. "Subtyping for Session Types in the π-Calculus". In: *Acta Informatica* 42.2–3 (2005), pp. 191–225.

[Ghi+19] Silvia Ghilezan et al. "Precise subtyping for synchronous multiparty sessions". In: *Journal of Logical and Algebraic Methods in Programming* 104 (2019), pp. 127 –173. ISSN: 2352-2208. DOI: `https://doi.org/10.1016/j.jlamp.2018.12.002`. URL: `http://www.sciencedirect.com/s`

[Ghi+20] Silvia Ghilezan et al. *Precise Subtyping for Asynchronous Multiparty Sessions*. 2020. arXiv: `2010.13925 [cs.LO]`.

[Hon93] Kohei Honda. “Types for Dyadic Interaction”. In: *4th International Conference on Concurrency Theory*. CONCUR’93. Springer LNCS 715, 1993, pp. 509–523.

[HVK98] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. “Language Primitives and Type Discipline for Structured Communication-Based Programming”. In: *7th European Symposium on Programming Languages and Systems (ESOP 1998)*. Springer LNCS 1381, 1998, pp. 122–138.

[KN19] Steve Klabnik and Carol Nichols. “The Rust Programming Language (Covers Rust 2018)”. In: No Starch Process, Aug. 2019. Chap. 16.

[MM84] Don P. Mitchell and Michael Merritt. “A Distributed Algorithm for Deadlock Detection and Resolution”. In: *Symposium on Principles of Distributed Computation (PODC 1984)*. ACM. Vancouver, British Columbia, Aug. 1984, pp. 282–284.

[PG15] Frank Pfenning and Dennis Griffith. “Polarized Substructural Session Types”. In: *Proceedings of the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2015)*. Ed. by A. Pitts. Invited talk. London, England: Springer LNCS 9034, Apr. 2015, pp. 3–22.

[PP19] Klaas Pruiksma and Frank Pfenning. “A Message-Passing Interpretation of Adjoint Logic”. In: *Workshop on Programming Language Approaches to Concurrency and Communication-Centric Software (PLACES)*. Ed. by F. Martins and D. Orchard. Prague, Czech Republic: EPTCS 291, Apr. 2019, pp. 60–79.

[San19] Chuta Sano. “On Session Typed Contracts for Imperative Languages”. Available as Technical Report CMU-CS-19-13. M.S. Thesis. Department of Computer Science, Carnegie Mellon University, Dec. 2019.

[San20] Ishani Santurkar. Personal communication. Oct. 2020.

[Ton15] Bernardo Toninho. “A Logical Foundation for Session-based Concurrent Computation”. Available as Technical Report CMU-CS-15-109. PhD thesis. Carnegie Mellon University and Universidade Nova de Lisboa, May 2015.

[Wad12] Philip Wadler. “Propositions as Sessions”. In: *Proceedings of the 17th International Conference on Functional Programming*. ICFP 2012. Copenhagen, Denmark: ACM Press, Sept. 2012, pp. 273–286.

[WPP17] Max Willsey, Rokhini Prabhu, and Frank Pfenning. “Design and Implementation of Concurrent C0”. In: *Electronic Proceedings in Theoretical Computer Science* 238 (Jan. 2017), pp. 73–82. DOI: 10.4204/EPTCS.238.8.

A Subtyping Relation

A.1 Subtyping of session types

As mentioned previously, the subtyping rules are defined coinductively.

$$\frac{}{1 \leq 1} \leq_1 \quad \frac{A_L \leq A'_L \qquad B_L \leq B'_L}{A_L \otimes B_L \leq A'_L \otimes B'_L} \leq_\otimes \quad \frac{A'_L \leq A_L \qquad B_L \leq B'_L}{A_L \multimap B_L \leq A'_L \multimap B'_L} \leq_\multimap$$

$$\frac{\forall i \in \overline{l} \quad A_{i_L} \leq A'_{i_L}}{\oplus\{\overline{l{:}A_L}\} \leq \oplus\{\overline{l{:}A'_L}, \overline{m{:}B_L}\}} \leq_\& \quad \frac{\forall i \in \overline{l} \quad A_{i_L} \leq A'_{i_L}}{\&\{\overline{l{:}A_L}, \overline{m{:}B_L}\} \leq \&\{\overline{l{:}A'_L}\}} \leq_\&$$

$$\frac{A_L \leq B_L}{\uparrow_L^S A_L \leq \uparrow_L^S B_L} \leq_{\uparrow_L^S} \quad \frac{A_L \leq B_L}{\uparrow_L^L A_L \leq \uparrow_L^L B_L} \leq_{\uparrow_L^L} \quad \frac{A_L \leq B_L}{\uparrow_L^S A_L \leq \uparrow_L^L B_L} \leq_{\uparrow_L^S \uparrow_L^L}$$

$$\frac{A_S \leq B_S}{\downarrow_L^S A_S \leq \downarrow_L^S B_S} \leq_{\downarrow_L^S} \quad \frac{A_L \leq B_L}{\downarrow_L^L A_L \leq \downarrow_L^L B_L} \leq_{\downarrow_L^L} \quad \frac{A_S \leq B_L}{\downarrow_L^S A_S \leq \downarrow_L^L B_L} \leq_{\downarrow_L^S \downarrow_L^L}$$

A.2 Lemmas

For the following lemmas, we will switch to a set-based formulation of subtyping: $A \leq B$ is represented as $(A, B) \in \mathrm{st}$

Lemma 1. *Subtyping is a partial order*

Proof. We want to show that

1. Reflexivity: $A \leq A$ for any A
2. Anti-symmetricity: $A \leq B \wedge B \leq A \implies A = B$
3. Transitivity: $A \leq B \wedge B \leq C \implies A \leq C$

All cases follow from coinduction on the subtyping rules.

□

A.2.1 Extended Types

Recall in the safe synchronization formulation, we introduce the extended (shared) types

$$\hat{A} \ ::= \ \bot \mid \top \mid A_S$$

where $\bot \leq A_S \leq \top$ for any A_S. The constants $\bot$ and $\top$ are compatible with our subtyping relation so far except we emphasize that $\bot$ and $\top$ must be taken to be the minimal and maximal shared session types and not session types in general. For example, it is not the case that $\bot \leq 1$ since 1 is a linear session type. However, it is the case that $\bot \leq \uparrow_L^L A_L$ by transitivity since $\uparrow_L^S A'_L \leq \uparrow_L^L A_L$ for some $A'_L \leq A_L$. In any case, when we fix our elements to constraints, we can define a meet semilattice with the meet operator $\wedge$.

The meet operator $\wedge$ is defined coinductively from the structure of its arguments. Note that there are many cases where these rules do not apply – in that case the result of the meet is $\bot$.

$$\begin{gathered}
1 \wedge 1 \rightarrow 1 \\
A_L \otimes A'_L \wedge B_L \otimes B'_L \rightarrow (A_L \wedge B_L) \otimes (A'_L \wedge B'_L) \\
A_L \multimap A'_L \wedge B_L \multimap B'_L \rightarrow (A_L \wedge B_L) \multimap (A'_L \wedge B'_L) \\
\&\{\overline{l{:}A_L}, \overline{m{:}B_L}\} \wedge \&\{\overline{l{:}A'_L}, \overline{n{:}C_L}\} \rightarrow \&\{\overline{l : (A_L \wedge A'_L)}, \overline{m{:}B_L}, \overline{n{:}C_L}\} \\
\oplus\{\overline{l{:}A_L}, \overline{m{:}B_L}\} \wedge \oplus\{\overline{l{:}A'_L}, \overline{n{:}C_L}\} \rightarrow \oplus\{\overline{l : (A_L \wedge A'_L)}\} \qquad (\bar{l} \text{ not empty}) \\
\uparrow_L^S A_L \wedge \uparrow_L^S B_L \rightarrow \uparrow_L^S (A_L \wedge B_L) \\
\uparrow_L^S A_L \wedge \uparrow_L^L B_L \rightarrow \uparrow_L^S (A_L \wedge B_S) \\
\uparrow_L^L A_L \wedge \uparrow_L^S B_L \rightarrow \uparrow_L^S (A_S \wedge B_L) \\
\uparrow_L^L A_L \wedge \uparrow_L^L B_L \rightarrow \uparrow_L^L (A_S \wedge B_S) \\
\downarrow_L^S A_S \wedge \downarrow_L^S B_S \rightarrow \downarrow_L^S (A_S \wedge B_S) \\
\downarrow_L^S A_S \wedge \downarrow_L^L B_L \rightarrow \downarrow_L^S (A_S \wedge B_L) \\
\downarrow_L^L A_L \wedge \downarrow_L^S B_S \rightarrow \downarrow_L^S (A_L \wedge B_S) \\
\downarrow_L^L A_L \wedge \downarrow_L^L B_L \rightarrow \downarrow_L^L (A_L \wedge B_L)
\end{gathered}$$

Intuitively, the idea with this construction is that on external choices, we take the union of the labels on both sides whereas on internal choices, we take the intersection of the labels on both sides. Since we do not allow the nullary internal choice $\oplus\{\}$ in the language, we require that the meet between two internal choices to be non-empty, that is, they must share at least one label. Otherwise, the meet construction should fail and therefore produce a $\bot$.

Lemma 2. *$\hat{A} \wedge \hat{B}$ is the greatest lower bound between $\hat{A}$ and $\hat{B}$.*

Proof. Commutativity, associativity, and idempotency follows from coinduction on the construction rules.
We next want to show that for any $\hat{A}$ and $\hat{B}$,

1. $\hat{A} \wedge \hat{B} \leq \hat{A}$
2. $\hat{A} \wedge \hat{B} \leq \hat{B}$

which again follows from coinduction. Intuitively, this is due to the the set of labels growing on external choices (union) and shrinking on internal choices (intersect), which directly align with the subtyping relation we desire.

The final condition is that for any $\hat{A}$ and $\hat{B}$, if there exists a $\hat{C}$ such that $\hat{C} \leq \hat{A}$ and $\hat{C} \leq \hat{B}$, then $\hat{C} \leq \hat{A} \wedge \hat{B}$. □

Remark We believe that a join operation can be similarly defined except we switch the union and intersects in the external and internal choices respectively while taking the smaller modal shifts instead of the bigger ones with its maximal type being $\top$. As of this moment, it is not clear whether this operator is immediately applicable, but we mention its existence to complete the lattice.

A.3 Subtyping of contexts

$$\frac{}{\cdot \leq \cdot}\ \leq_{\cdot} \qquad \frac{\Delta \leq \Delta' \quad A_L \leq A'_L}{\Delta, x_L{:}A_L \leq \Delta', x_L{:}A'_L}\ \leq_{\Delta}$$

$$\frac{}{\Gamma \preceq \cdot}\ \preceq_{\cdot} \qquad \frac{\Gamma \preceq \Gamma' \quad \hat{A} \leq \hat{A}'}{\Gamma, x_S{:}\hat{A} \preceq \Gamma', x_S{:}\hat{A}'}\ \preceq_{\Gamma}$$

Lemma 3. *Type judgments are stable under a smaller shared context.
Let $\Gamma' \preceq \Gamma$, then $\Gamma; \Delta \vdash P :: (z_L{:}C_L) \rightarrow \Gamma'; \Delta \vdash P :: (z_L{:}C_L)$.*

Proof. We first prove the admissibility of the substitution of a shared channel by a smaller type in a typing judgment. In particular, we will begin by showing that if $\Gamma, x_S{:}\hat{A}; \Delta \vdash P :: (z_L{:}C_L)$, then $\Gamma, x_S{:}\hat{B}; \Delta \vdash P :: (z_L{:}C_L)$ for some $\hat{B} \leq \hat{A}$ by induction on the derivation of $\Gamma, x_S{:}\hat{A}; \Delta \vdash P :: (z_L{:}C_L)$.

First, we begin by pointing out that rules that do not use a_S (most of them) are trivial since we can just appeal to the induction hypothesis on the premise(s) in the appropriate derivation. The rules that can use a_S are $ID_S, ID_{LS}, SP_{LL}, SP_{LS}, SP_{SS}, \uparrow_L^S L, \multimap L_S$, and $\otimes R_S$. For these cases, we can confirm that the substitution is valid by using the induction hypothesis and using transitivity of $\leq$. We will present one such case:

Case 1.

$$\frac{\hat{A} \leq A_L \quad \Gamma, x_S{:}\hat{A}; \Delta \vdash P :: (y_L{:}B_L)}{\Gamma, x_S{:}\hat{A}; \Delta \vdash \text{send } y_L\ x_S; P :: (y_L{:}A_L \otimes B_L)} \otimes R_S$$

Then by induction hypothesis, $\Gamma, x_S{:}\hat{B}; \Delta \vdash P :: (y_L{:}B_L)$. Furthermore, by transitivity, $\hat{B} \leq A_L$. Therefore by $\otimes R_S$, $\Gamma, x_S{:}\hat{A}; \Delta \vdash \text{send } y_L\ x_S; P :: (y_L{:}A_L \otimes B_L)$

After showing that substitution by a smaller type in the shared context Γ is admissible, we proceed by induction on $\Gamma' \preceq \Gamma$.

Case 1.

$$\frac{}{\Gamma' \preceq \cdot} \preceq_{\cdot}$$

Then we have $\cdot; \Delta \vdash P :: (z_L{:}C_L)$. By weakening, we conclude $\Gamma'; \Delta \vdash P :: (z_L{:}C_L)$

Case 2.

$$\frac{\Gamma \preceq \Gamma' \quad \hat{A} \leq \hat{A}'}{\Gamma, x_S{:}\hat{A} \preceq \Gamma', x_S{:}\hat{A}'} \preceq_\Gamma$$

Then we have $\Gamma, x_S{:}\hat{A}; \Delta \vdash P :: (z_L{:}C_L)$. By induction hypothesis, we have $\Gamma, x_S{:}\hat{A}; \Delta \vdash P :: (z_L{:}C_L)$. Then, since $\hat{A}' \leq \hat{A}$, we have $\Gamma, x_S{:}\hat{A}'; \Delta \vdash P :: (z_L{:}C_L)$ by previously proved admissibility of substitution by smaller type for shared context.

□

B Configuration

B.1 Definitions

A configuration consists of a list of shared processes Λ and a list of linear processes Θ. The order of shared processes have no structure, but the order of linear processes can be seen to form a tree structure; a linear process can use channels offered by processes to its right, and due to linearity, if it is using a channel, it must be the unique process doing so.

$$\begin{aligned}
\Omega &::= \Lambda ; \Theta \\
\Lambda &::= \cdot \mid \Lambda_1, \Lambda_2 \mid \text{proc}(a_S, P) \mid \text{unavail}(a_S) \\
\Theta &::= \cdot \mid \text{proc}(a_L, P), \Theta' \mid \text{connect}(a_L, b_S), \Theta'
\end{aligned}$$

Well-formedness Λ is well-formed if for any channel name a, $\text{proc}(a_S, P), \text{unavail}(a_S) \notin \Lambda$. Similarly, Θ is well-formed if for any a, $\Psi_a, \Psi'_a \notin \Theta$. The configuration $\Lambda;\Theta$ is well-formed if $\Psi_a \in \Theta \rightarrow \text{unavail}(a_S) \in \Lambda$.

B.2 Configuration Typing

A well-formed configuration $\Lambda;\Theta$ is typed by its shared and linear fragments.

$$\frac{\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta :: (\Delta)}{\Gamma \models \Lambda;\Theta :: (\Gamma;\Delta)}\ \Omega$$

B.2.1 Shared Fragment Λ

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Lambda 1 \qquad \frac{\Gamma \models \Lambda_1 :: (\Gamma_1) \quad \Gamma \models \Lambda_2 :: (\Gamma_2)}{\Gamma \models \Lambda_1, \Lambda_2 :: (\Gamma_1, \Gamma_2)}\ \Lambda 2$$

$$\frac{\vdash (A'_S, A_S, \top)\ \text{dfsync} \quad \Gamma \vdash P :: (a_S{:}A'_S)}{\Gamma \models \text{proc}(a_S, P) :: (a_S{:}A_S)}\ \Lambda 3 \qquad \frac{}{\Gamma \models \text{unavail}(a_S) :: (a_S{:}\hat{A})}\ \Lambda 4$$

B.2.2 Linear Fragment Θ

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Theta 1 \qquad \frac{b_S{:}\hat{B} \in \Gamma \quad b_S \leq A_L \quad \Gamma \models \Theta' :: (\Delta')}{\Gamma \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, \Delta')}\ \Theta 2$$

$$\frac{a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync} \quad \Gamma; \Delta_a \vdash P :: (a_L{:}A'_L) \quad \Gamma \models \Theta' :: (\Delta_a, \Delta')}{\Gamma \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')}\ \Theta 3$$

B.3 Meta Theory

Lemma 4. *If* $\Gamma \models \Psi, \Theta :: (\Delta)$*, then* $\Gamma \models \Theta :: (\Delta')$ *for some* Δ'*.*
More generally, if $\Gamma \models \Theta_1, \Theta_2 :: (\Delta)$*, then* $\Gamma \models \Theta_2 :: (\Delta')$ *for some* Δ'*.*

Proof. For the first part, by case analysis on the derivation of $\Gamma \models \Psi, \Theta :: (\Delta)$. In both cases ($\Theta$2 and Θ3), we directly see that $\Gamma \models \Theta :: (\Delta')$ for some Δ'.
For the second part, we can repeatedly apply the first part sequentially for every $\Psi \in \Theta_1$. □

Lemma 5. *If* $\Gamma \models \Psi, \Theta :: (\Delta)$*,* $\Gamma \models \Theta :: (\Delta_p)$*, and* $\Gamma \models \Theta' :: (\Delta_p)$*, then*

$$\Gamma \models \Psi, \Theta' :: (\Delta)$$

More generally, if $\Gamma \models \Theta_1, \Theta_2 :: (\Delta)$*,* $\Gamma \models \Theta_2 :: (\Delta_p)$*, and* $\Gamma \models \Theta_2' :: (\Delta_p)$*, then*

$$\Gamma \models \Theta_1, \Theta_2 :: (a_L{:}A_L, \Delta)$$

Proof. For the first part, by case analysis on the derivation of $\Gamma \models \Psi, \Theta :: (\Delta)$. In both cases ($\Theta$2 and Θ3), we can directly substitute Θ' for Θ where it appears in the configuration judgment.
For the second part, we can repeatedly apply the first part sequentially for every $\Psi \in \Theta_1$. □

Remark It turns out that we can prove a stronger property that we can replace Θ with Θ' that offers a subtype of what Θ offers, but we will only need this weaker lemma that requires the offering types to be equivalent for this paper.

Lemma 6. *If* $\Gamma \models \Psi_a, \Theta_1, \Psi_b, \Theta_2 :: (a_L{:}A_L, \Delta)$ *and* Ψ_a *uses* b_L*, then*

$$\Gamma \models \Psi_a, \Psi_b, \Theta_1, \Theta_2 :: (a_L{:}A_L, \Delta)$$

Proof. By well-formedness, Ψ_b is the only process in the configuration offering b_L. Furthermore by linearity, there can only be one process that use b_L, which is Ψ_a by assumption, so b_L will not be consumed by any processes in Θ_1. Therefore, we can repeatedly move Ψ_b to the left in the configuration until it is to the right of Ψ_a, the unique process using b_L. □

Lemma 7. *If* $\Gamma \models \Psi_a, \Theta' :: (a_L{:}A_L', \Delta')$*, then for any* B_L *such that* $A_L' \leq B_L$*,* $\Gamma \models \Psi_a, \Theta' :: (a_L{:}B_L, \Delta')$*.*

Proof. By inversion on the derivation of $\Gamma \models \Psi_a, \Theta' :: (a_L{:}A_L', \Delta)$.

Case 1.

$$\frac{b_S{:}\hat{B} \in \Gamma \quad \hat{b} \leq A_L \quad \Gamma \models \Theta' :: (\Delta')}{\Gamma \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, a_L{:}A'_L, \Delta')}\ \Theta 2$$

By transitivity, $\hat{B} \leq B_L$ therefore $\Gamma \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, a_L{:}B_L, \Delta')$
Case 2.

$$\frac{a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync} \quad \Gamma; \Delta_a \vdash P :: (a_L{:}A'_L) \quad \Gamma \models \Theta' :: (\Delta_a, \Delta')}{\Gamma \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')}\ \Theta 3$$

By transitivity, $A'_L \leq B_L$ and therefore $\vdash (A'_L, B_L, \hat{A})$ dfsync by Lemma 10.
Therefore, $\Gamma \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')$

□

Lemma 8. *Substitution by a smaller shared context is admissible. Let $\Gamma' \preceq \Gamma$ then*

1. *If $\Gamma \models \Theta :: (\Delta)$ for some Θ, Δ, then $\Gamma' \models \Theta :: (\Delta)$*
2. *If $\Gamma \models \Lambda :: (\Gamma'')$ for some Λ, Γ'', then $\Gamma' \models \Theta :: (\Gamma'')$*

Proof. For the first part, by induction on the derivation of $\Gamma \models \Theta :: (\Delta)$.
Case 1.

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Theta 1$$

Any Γ applies, so in particular any $\Gamma' \preceq \Gamma$ will as well.
Case 2.

$$\frac{b_S{:}\hat{B} \in \Gamma \quad \hat{B} \leq A_L \quad \Gamma \models \Theta' :: (\Delta')}{\Gamma \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, \Delta')}\ \Theta 2$$

By exchange, we can assume without loss of generality that $\Gamma = b_S{:}\hat{B}, \Gamma_r$. Similarly, we can assume without loss of generality that $\Gamma' = b_S{:}\hat{B}', \Gamma'_r$ where $\hat{B}' \leq \hat{B}$ and $\Gamma'_r \preceq \Gamma$.
$\hat{B}' \leq A_L$ follows by transitivity of $\leq$ and $\Gamma' \models \Theta' :: (\Delta_a, \Delta')$ follows from induction hypothesis. Therefore,

$$\Gamma' \models \text{connect}(a_L, b_S), \Theta' :: (a : A_L, \Delta')$$

Case 3.

$$\frac{a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync} \quad \Gamma; \Delta_a \vdash P :: (a_L{:}A'_L) \quad \Gamma \models \Theta' :: (\Delta_a, \Delta')}{\Gamma \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')}\ \Theta 3$$

By exchange, we can assume without loss of generality that $\Gamma = a_S{:}\hat{A}, \Gamma_r$. Similarly, we can assume without loss of generality that $\Gamma' = a_S{:}\hat{A}', \Gamma'_r$ where $\hat{B}' \leq \hat{B}$ and $\Gamma'_r \preceq \Gamma$.
$\vdash (A'_L, A_L, \hat{A}')$ dfsync follows from Lemma 12, $\Gamma'; \Delta_a \vdash P :: (a_L{:}A'_L)$ follows from Lemma 3, and
$\Gamma' \models \Theta' :: (\Delta_a, \Delta')$ follows from induction hypothesis. Therefore,

$$\Gamma' \models \text{proc}(a_L, P), \Theta' :: (a : A_L, \Delta')$$

For the second part, by induction on the derivation of $\Gamma \models \Lambda :: (\Delta')$

Case 1.

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Lambda 1$$

Any Γ applies, so in particular any $\Gamma' \preceq \Gamma$ will as well.

Case 2.

$$\frac{\Gamma \models \Lambda_1 :: (\Gamma_1) \quad \Gamma \models \Lambda_2 :: (\Gamma_2)}{\Gamma \models \Lambda_1, \Lambda_2 :: (\Gamma_1, \Gamma_2)}\ \Lambda 2$$

Both $\Gamma' \models \Lambda_1 :: (\Gamma_1)$ and $\Gamma' \models \Lambda_2 :: (\Gamma_2)$ follow from induction hypothesis. Therefore,

$$\Gamma' \models \Lambda_1, \Lambda_2 :: (\Gamma_1, \Gamma_2)$$

Case 3.

$$\frac{\vdash (A'_S, A_S, \top)\ \text{dfsync} \quad \Gamma \vdash P :: (a_S{:}A'_S)}{\Gamma \models \text{proc}(a_S, P) :: (a_S{:}A_S)}\ \Lambda 3$$

$\Gamma' \vdash P :: (a_S{:}A'_S)$ follows from Lemma 3. Therefore,

$$\Gamma \models \text{proc}(a_S, P) :: (a_S{:}A_S)$$

Case 4.

$$\frac{}{\Gamma \models \text{unavail}(a_S) :: (a_S{:}\hat{A})}\ \Lambda 4$$

Any Γ applies, so in particular any $\Gamma' \preceq \Gamma$ will as well.

□

Lemma 9. *Given a well-formed* $\Lambda; \Theta$*,* $\forall \mathit{proc}(a_S, -) \in \Lambda, \Psi_a \notin \Theta$

Proof. By well-formedness of Λ, $\text{proc}(a_S, -) \in \Lambda$ means that $\text{unavail}(a_S) \notin \Lambda$. By the contrapositive of well-formedness of $\Lambda; \Theta$, $\text{unavail}(a_S) \notin \Lambda \implies \Psi_a \notin \Theta$ □

C Safely Synchronizing

C.1 Rules

As introduced previously, the safely synchronizing judgment rules are defined coinductively.

$$\frac{}{\vdash (1, 1, \hat{D})\ \text{dfsync}}\ D1 \qquad \frac{\vdash (B_L, B'_L, \hat{D})\ \text{dfsync}}{\vdash (A_L \otimes B_L, A'_L \otimes B'_L, \hat{D})\ \text{dfsync}}\ D\otimes \qquad \frac{\vdash (B_L, B'_L, \hat{D})\ \text{dfsync}}{\vdash (A_L \multimap B_L, A'_L \multimap B'_L, \hat{D})\ \text{dfsync}}\ D{\multimap}$$

$$\frac{\forall i \in \overline{l} \quad \vdash (A_{i_L}, A'_{i_L}, \hat{D}) \text{ dfsync}}{\vdash (\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}A'_L}, \overline{m{:}B_L}\}, \hat{D}) \text{ dfsync}}\ D\oplus \qquad \frac{\forall i \in \overline{l} \quad \vdash (A_{i_L}, A'_{i_L}, \hat{D}) \text{ dfsync}}{\vdash (\&\{\overline{l{:}A_L}, \overline{m{:}B_L}\}, \&\{\overline{l{:}A'_L}\}, \hat{D}) \text{ dfsync}}\ D\&$$

$$\frac{\vdash (A_L, A'_L, \hat{D}) \text{ dfsync}}{\vdash (\uparrow_L^L A_L, \uparrow_L^L A'_L, \hat{D}) \text{ dfsync}}\ D\uparrow_L^L \qquad \frac{\vdash (A_L, A'_L, \hat{D}) \text{ dfsync}}{\vdash (\downarrow_L^L A_L, \downarrow_L^L A'_L, \hat{D}) \text{ dfsync}}\ D\downarrow_L^L$$

$$\frac{\vdash (A_L, A'_L, \uparrow_L^S A_L) \text{ dfsync}}{\vdash (\uparrow_L^S A_L, \uparrow_L^S A'_L, \top) \text{ dfsync}}\ D\uparrow_L^S \qquad \frac{\vdash (A_S, A'_S, \top) \text{ dfsync} \quad \downarrow_L^S A_S \leq \hat{D}}{\vdash (\downarrow_L^S A_S, \downarrow_L^S A'_S, \hat{D}) \text{ dfsync}}\ D\downarrow_L^S$$

$$\frac{\vdash (A_L, A'_L, \uparrow_L^S A_L) \text{ dfsync}}{\vdash (\uparrow_L^S A_L, \uparrow_L^L A'_L, \top) \text{ dfsync}}\ D\uparrow_L^S\uparrow_L^L \qquad \frac{\vdash (A_S, A'_L, \top) \text{ dfsync} \quad \downarrow_L^S A_S \leq \hat{D}}{\vdash (\downarrow_L^S A_S, \downarrow_L^L A'_L, \hat{D}) \text{ dfsync}}\ D\downarrow_L^S\downarrow_L^L$$

C.2 Lemmas

To prove the following lemmas, we switch to a set-based formulation of safe synchronization; $\vdash (A, B, \hat{D})$ dfsync is written as $(A, B, \hat{D}) \in$ dfsync. We also define a monotone map F from the coinductive definition of dfsync, giving us dfsync $\in F$(dfsync); that is, dfsync is F-consistent.

The first lemma says that we can replace the second argument with a bigger type:

Lemma 10. *If $A \leq B \leq C$ with all same modalities (that is, A, B, C are either all linear or all shared) and $\vdash (A, B, \hat{D})$ dfsync, then $\vdash (A, C, \hat{D})$ dfsync for some $\hat{D}$.*

Proof. We want to show that

$$\text{dfsync}' \ ::= \ \text{dfsync} \cup \text{dfsync}_{\Uparrow}$$

is F-consistent with

$$\text{dfsync}_{\Uparrow} \ ::= \ \{(A, C, \hat{D}) \mid \exists B. B \leq C \wedge (A, B, \hat{D}) \in \text{dfsync}\}$$

Again, where A, B, C must all be of the same modality.
We will prove F-consistency of dfsync$'$, that is, dfsync$' \in F$(dfsync$'$) by showing that each of the two sets dfsync and dfsync$_{\Uparrow}$ are subsets of F(dfsync$'$).

First, dfsync $\subseteq F$(dfsync$'$) immediately follows because dfsync $\subseteq F$(dfsync) and F(dfsync) $\subseteq F$(dfsync$'$) by monotonicity of F given dfsync $\subseteq$ dfsync$'$.

We will now consider dfsync$_{\Uparrow} \in F$(dfsync$'$) by case analysis on the structure of A. We can uniquely infer the structure of B and C from the structure of A by inversion on the appropriate subtyping rule for most cases.

Case 1. $A = \uparrow_L^L A'_L$; then $B = \uparrow_L^L B'_L$ and $C = \uparrow_L^L C'_L$ with $A'_L \le B'_L \le C'_L$.

$$\begin{aligned}
(\uparrow_L^L A'_L, \uparrow_L^L C'_L, \hat{D}) \in \text{dfsync}_{\Uparrow}, (\uparrow_L^L A'_L, \uparrow_L^L B'_L, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
(A'_L, B'_L, \hat{D}) \in \text{dfsync} && \text{(by inversion on } D\uparrow_L^L)\\
(A'_L, C'_L, \hat{D}) \in \text{dfsync}_{\Uparrow} && \text{(by definition of dfsync}_{\Uparrow} \text{ with } B'_L \le C'_L)\\
(A'_L, C'_L, \hat{D}) \in \text{dfsync}' && \text{(since dfsync}_{\Uparrow} \subseteq \text{dfsync}')\\
(\uparrow_L^L A'_L, \uparrow_L^L C'_L, \hat{D}) \in F(\text{dfsync}') && \text{(by } D\uparrow_L^L)
\end{aligned}$$

$\downarrow_L^L, \otimes$, and $\multimap$ follow a similar pattern of appealing to the covariance of subtyping on the continuation types.

Case 2. $A = \oplus\{\overline{l{:}A_L}\}$; then $B = \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}$ and $C = \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}$ with $A_{i_L} \le B_{i_L} \le C_{i_L}\ \forall i \in \overline{l}$ and $B_{i_L} \le C_{i_L}\ \forall i \in \overline{m}$.

$$\begin{aligned}
(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}, \hat{D}) \in \text{dfsync}_{\Uparrow} && \\
(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
(\forall i \in \overline{l})\ (A_{i_L}, B_{i_L}, \hat{D}) \in \text{dfsync} && \text{(by inversion on } D\oplus)\\
(\forall i \in \overline{l})\ (A_{i_L}, C_{i_L}, \hat{D}) \in \text{dfsync}_{\Uparrow} && \\
&& \text{(by definition of dfsync}_{\Uparrow} \text{ with } B_{i_L} \le C_{i_L})\\
(\forall i \in \overline{l})\ (A_{i_L}, C_{i_L}, \hat{D}) \in \text{dfsync}' && \text{(since dfsync}_{\Uparrow} \subseteq \text{dfsync}')\\
(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}, \hat{D}) \in F(\text{dfsync}') && \text{(by } D\oplus)
\end{aligned}$$

$D\&$ follows a similar pattern.

Case 3. $A = \downarrow_L^S A_S$; then there are three possible assignments to B and C that satisfies the subtyping constraints, so we will continue by subcasing on the structure of B and C.

Subcase 1. $B = \downarrow_L^S B_S$ and $C = \downarrow_L^S C_S$ with $A_S \le B_S \le C_S$.

$$\begin{aligned}
(\downarrow_L^S A_S, \downarrow_L^S C_S, \hat{D}) \in \text{dfsync}_{\Uparrow}, (\downarrow_L^S A_S, \downarrow_L^S B_S, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
(A_S, B_S, \top) \in \text{dfsync}, A_S \le \hat{D} && \text{(by inversion on } D\downarrow_L^S)\\
(A_S, C_S, \top) \in \text{dfsync}_{\Uparrow} && \text{(by definition of dfsync}_{\Uparrow} \text{ with } B_S \le C_S)\\
(A_S, C_S, \top) \in \text{dfsync}' && \text{(since dfsync}_{\Uparrow} \subseteq \text{dfsync}')\\
(\downarrow_L^S A_S, \downarrow_L^S C_S, \hat{D}) \in F(\text{dfsync}') && \text{(by } D\downarrow_L^S)
\end{aligned}$$

Subcase 2. $B = {\downarrow}^S_L B_S$ and $C = {\downarrow}^L_L C_L$ with $A_S \leq B_S \leq C_L$.

$$\begin{aligned}
({\downarrow}^S_L A_S, {\downarrow}^L_L C_L, \hat{D}) &\in \mathrm{dfsync}_{\Uparrow}, ({\downarrow}^S_L A_S, {\downarrow}^S_L B_S, \hat{D}) \in \mathrm{dfsync} && \text{(this case)}\\
(A_S, B_S, \top) &\in \mathrm{dfsync}, A_S \leq \hat{D} && \text{(by inversion on } D{\downarrow}^S_L\text{)}\\
(A_S, C_L, \top) &\in \mathrm{dfsync}_{\Uparrow} && \text{(by definition of } \mathrm{dfsync}_{\Uparrow} \text{ with } B_S \leq C_L\text{)}\\
(A_S, C_L, \top) &\in \mathrm{dfsync}' && \text{(since } \mathrm{dfsync}_{\Uparrow} \subseteq \mathrm{dfsync}'\text{)}\\
({\downarrow}^S_L A_S, {\downarrow}^L_L C_L, \hat{D}) &\in F(\mathrm{dfsync}') && \text{(by } D{\downarrow}^S_L\text{)}
\end{aligned}$$

Subcase 3. $B = {\downarrow}^L_L B_L$ and $C = {\downarrow}^L_L C_L$ with $A_S \leq B_L \leq C_L$.

$$\begin{aligned}
({\downarrow}^S_L A_S, {\downarrow}^L_L C_L, \hat{D}) &\in \mathrm{dfsync}_{\Uparrow}, ({\downarrow}^S_L A_S, {\downarrow}^L_L B_L, \hat{D}) \in \mathrm{dfsync} && \text{(this case)}\\
(A_S, B_L, \top) &\in \mathrm{dfsync}, A_S \leq \hat{D} && \text{(by inversion on } D{\downarrow}^S_L{\downarrow}^L_L\text{)}\\
(A_S, C_L, \top) &\in \mathrm{dfsync}_{\Uparrow} && \text{(by definition of } \mathrm{dfsync}_{\Uparrow} \text{ with } B_L \leq C_L\text{)}\\
(A_S, C_L, \top) &\in \mathrm{dfsync}' && \text{(since } \mathrm{dfsync}_{\Uparrow} \subseteq \mathrm{dfsync}'\text{)}\\
({\downarrow}^S_L A_S, {\downarrow}^L_L C_L, \hat{D}) &\in F(\mathrm{dfsync}') && \text{(by } D{\downarrow}^S_L\text{)}
\end{aligned}$$

Case 4. $A = {\uparrow}^S_L A_L$; then there are three possible assignments to B and C that satisfies the subtyping constraints, so we will continue by subcasing on the structure of B and C.
Subcase 1. $B = {\uparrow}^S_L B_L$ and $C = {\uparrow}^S_L C_L$ with $A_L \leq B_L \leq C_L$.

$$\begin{aligned}
({\uparrow}^S_L A_L, {\uparrow}^S_L C_L, \top) &\in \mathrm{dfsync}_{\Uparrow}, ({\uparrow}^S_L A_L, {\uparrow}^S_L B_L, \top) \in \mathrm{dfsync} && \text{(this case)}\\
(A_L, B_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync} && \text{(by inversion on } D{\uparrow}^S_L\text{)}\\
(A_L, C_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync}_{\Uparrow} && \text{(by definition of } \mathrm{dfsync}_{\Uparrow} \text{ with } A_L \leq B_L\text{)}\\
(A_L, C_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync}' && \text{(since } \mathrm{dfsync}_{\Uparrow} \subseteq \mathrm{dfsync}'\text{)}\\
({\uparrow}^S_L A_L, {\uparrow}^S_L C_L, \top) &\in F(\mathrm{dfsync}') && \text{(by } D{\uparrow}^S_L\text{)}
\end{aligned}$$

Subcase 2. $B = {\uparrow}^S_L B_L$ and $C = {\uparrow}^L_L C_L$ with $A_L \leq B_L \leq C_L$.

$$\begin{aligned}
({\uparrow}^S_L A_L, {\uparrow}^L_L C_L, \top) &\in \mathrm{dfsync}_{\Uparrow}, ({\uparrow}^S_L A_L, {\uparrow}^S_L B_L, \top) \in \mathrm{dfsync} && \text{(this case)}\\
(A_L, B_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync} && \text{(by inversion on } D{\uparrow}^S_L\text{)}\\
(A_L, C_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync}_{\Uparrow} && \text{(by definition of } \mathrm{dfsync}_{\Uparrow} \text{ with } A_L \leq B_L\text{)}\\
(A_L, C_L, {\uparrow}^S_L A_L) &\in \mathrm{dfsync}' && \text{(since } \mathrm{dfsync}_{\Uparrow} \subseteq \mathrm{dfsync}'\text{)}\\
({\uparrow}^S_L A_L, {\uparrow}^L_L C_L, \top) &\in F(\mathrm{dfsync}') && \text{(by } D{\uparrow}^S_L\text{)}
\end{aligned}$$

Subcase 3. $B = \uparrow_L^L B_L$ and $C = \uparrow_L^L C_L$ with $A_L \leq B_L \leq C_L$.

$$\begin{array}{lr}
(\uparrow_L^S A_L, \uparrow_L^L C_L, \top) \in \text{dfsync}_{\Uparrow}, (\uparrow_L^S A_L, \uparrow_L^L B_L, \top) \in \text{dfsync} & \text{(this case)} \\
(A_L, B_L, \uparrow_L^S A_L) \in \text{dfsync} & \text{(by inversion on } D\uparrow_L^S\uparrow_L^L) \\
(A_L, C_L, \uparrow_L^S A_L) \in \text{dfsync}_{\Uparrow} & \text{(by definition of dfsync}_{\Uparrow} \text{ with } A_L \leq B_L) \\
(A_L, C_L, \uparrow_L^S A_L) \in \text{dfsync}' & \text{(since dfsync}_{\Uparrow} \subseteq \text{dfsync}') \\
(\uparrow_L^S A_L, \uparrow_L^L C_L, \top) \in F(\text{dfsync}') & \text{(by } D\uparrow_L^S)
\end{array}$$

We missed one case, when $A = B = C = 1$, but this case is trivial since $\text{dfsync}_{\Uparrow}$ does not add any new members to the set.

□

The next lemma says that we can replace the first argument with a smaller type if the result is already safely synchronizing with some other constraint:

Lemma 11. *If $A \leq B \leq C$ with all same modalities, $\vdash (B, C, \hat{D})$ dfsync, and $\vdash (A, C, \hat{E})$ dfsync, then $\vdash (A, C, \hat{D})$ dfsync for some $\hat{D}$ and $\hat{E}$.*

Proof. We want to show that

$$\text{dfsync}' \ ::= \ \text{dfsync} \cup \text{dfsync}_{\Downarrow}$$

is F-consistent with

$$\text{dfsync}_{\Downarrow} \ ::= \ \{(A, C, \hat{D}) \mid \exists B. A \leq B \wedge (B, C, \hat{D}) \in \text{dfsync} \wedge \exists \hat{E}. (A, C, \hat{E}) \in \text{dfsync}\}$$

The proof is very similar in style to the previous lemma, but there is one additional constraint that $(A, C, \hat{E}) \in$ dfsync for any constraint $\hat{E}$. This assumption is only necessary for the $\uparrow_L^S$ case.
In any case, we will prove F-consistency of dfsync$'$, that is, dfsync$' \in F(\text{dfsync}')$ by showing that each of the three sets dfsync and $\text{dfsync}_{\Downarrow}$ are subsets of $F(\text{dfsync}')$.

First, dfsync $\subseteq F(\text{dfsync}')$ immediately follows from the same argument as in the previous proof.

We will now consider $\text{dfsync}_{\Downarrow} \in F(\text{dfsync}')$ by case analysis on the structure of B. Because all of A, B, C have the same modality, we can uniquely infer the structure of A and C from the structure of B by inversion on the appropriate subtyping rule.

Case 1. $A = \uparrow_L^L A'_L$; then $B = \uparrow_L^L B'_L$ and $C = \uparrow_L^L C'_L$ with $A'_L \leq B'_L \leq C'_L$.

$$
\begin{array}{rlr}
(\uparrow_L^L A'_L, \uparrow_L^L C'_L, \hat{D}) \in \text{dfsync}_{\Downarrow}, (\uparrow_L^L B'_L, \uparrow_L^L C'_L, \hat{D}) \in \text{dfsync} & & \text{(this case)} \\
(B'_L, C'_L, \hat{D}) \in \text{dfsync} & & \text{(by inversion on } D\uparrow_L^L) \\
(A'_L, C'_L, \hat{D}) \in \text{dfsync}_{\Downarrow} & & \text{(by definition of dfsync}_{\Downarrow} \text{ with } A'_L \leq B'_L) \\
(A'_L, C'_L, \hat{D}) \in \text{dfsync}' & & \text{(since dfsync}_{\Downarrow} \subseteq \text{dfsync}') \\
(\uparrow_L^L A'_L, \uparrow_L^L C'_L, \hat{D}) \in F(\text{dfsync}') & & \text{(by } D\uparrow_L^L)
\end{array}
$$

$\downarrow_L^L$, $\otimes$, and $\multimap$ follow a similar pattern of appealing to the covariance of subtyping on the continuation types.

Case 2. $A = \oplus\{\overline{l{:}A_L}\}$; then $B = \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}$ and $C = \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}$ with $A_{iL} \leq B_{iL} \leq C_{iL}\ \forall i \in \overline{l}$ and $B_{iL} \leq C_{iL}\ \forall i \in \overline{m}$.

$$
\begin{array}{rr}
(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}, \hat{D}) \in \text{dfsync}_{\Downarrow} & \\
(\oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}, \hat{D}) \in \text{dfsync} & \text{(this case)} \\
(\forall i \in \overline{l}, \overline{m})\ (B_{iL}, C_{iL}, \hat{D}) \in \text{dfsync} & \text{(by inversion on } D\oplus) \\
(\forall i \in \overline{l})\ (A_{iL}, C_{iL}, \hat{D}) \in \text{dfsync}_{\Downarrow} & \\
 & \text{(by definition of dfsync}_{\Downarrow} \text{ with } A_{iL} \leq B_{iL}) \\
(\forall i \in \overline{l})\ (A_{iL}, C_{iL}, \hat{D}) \in \text{dfsync}' & \text{(since dfsync}_{\Downarrow} \subseteq \text{dfsync}') \\
(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}C_L}, \overline{m{:}C_L}, \overline{n{:}C_L}\}, \hat{D}) \in F(\text{dfsync}') & \text{(by } D\oplus)
\end{array}
$$

$D\&$ follows a similar pattern.

Case 3. $A = \downarrow_L^S A_S$; similar to the proof of Lemma 10, there are three possible assignments for B and C. We will present one of those subcases: let $B = \downarrow_L^S B_S$ and $C = \downarrow_L^S C_S$ with $A_S \leq B_S \leq C_S$. The other two cases are similar.

$$
\begin{array}{rr}
(\downarrow_L^S A_S, \downarrow_L^S C_S, \hat{D}) \in \text{dfsync}_{\Downarrow}, (\downarrow_L^S B_S, \downarrow_L^S C_S, \hat{D}) \in \text{dfsync} & \text{(this case)} \\
(B_S, C_S, \top) \in \text{dfsync}, B_S \leq \hat{D} & \text{(by inversion on } D\downarrow_L^S) \\
(A_S, C_S, \top) \in \text{dfsync}_{\Downarrow} & \text{(by definition of dfsync}_{\Downarrow} \text{ with } A_S \leq B_S) \\
(A_S, C_S, \top) \in \text{dfsync}' & \text{(since dfsync}_{\Downarrow} \subseteq \text{dfsync}') \\
A_S \leq \hat{D} & \text{(because } A_S \leq B_S \leq \hat{D}) \\
(\downarrow_L^S A_S, \downarrow_L^S C_S, \hat{D}) \in F\text{dfsync}' & \text{(by } D\downarrow_L^S)
\end{array}
$$

Case 4. $A = \uparrow_L^S A_L$; again, there are three possible assignments for B and C, and we will take the subcase when $B = \uparrow_L^S B_L$ and $C = \uparrow_L^S C_L$ with $A_L \leq B_L \leq C_L$. The other two cases are similar. This case finally uses our assumption that $(A, C, \hat{E}) \in \text{dfsync}$ – $\hat{E}$ must be $\top$ due to $A = \uparrow_L^S A_L$.

$$\begin{aligned}
(\uparrow_L^S A_L, \uparrow_L^S C_L, \top) &\in \text{dfsync}_{\Downarrow}, (\uparrow_L^S B_L, \uparrow_L^S C_L, \top) \in \text{dfsync} && \text{(this case)}\\
(\uparrow_L^S A_L, \uparrow_L^S C_L, \top) &\in \text{dfsync} && \text{(By assumption with } \hat{E} = \top)\\
(A_L, C_L, \uparrow_L^S A_L) &\in \text{dfsync} && \text{(by inversion on } D{\uparrow_L^S})\\
(A_L, C_L, \uparrow_L^S A_L) &\in \text{dfsync}' && \text{(since dfsync} \subseteq \text{dfsync}')\\
(\uparrow_L^S A_L, \uparrow_L^S C_L, \top) &\in F(\text{dfsync}') && \text{(by } D{\uparrow_L^S})
\end{aligned}$$

We missed one case, when $A = B = C = 1$, but this case is trivial since $\text{dfsync}_{\Downarrow}$ does not add any new members to the set.

□

The next lemma allows us to replace the third argument (the constraint) with a smaller one if the first argument is linear:

Lemma 12. *If* $\vdash (A_L, B_L, \hat{C})$ *dfsync and* $\hat{D} \leq \hat{C}$, *then* $\vdash (A_L, B_L, \hat{D})$ *dfsync for some* $A_L, B_L, \hat{C}$, *and* $\hat{D}$.

Proof. By coinduction where $\text{dfsync}' = \text{dfsync} \cup \text{dfsync}_{\hat{\Downarrow}}$ such that

$$\text{dfsync}_{\hat{\Downarrow}} = \{(A_L, B_L, \hat{D}) \mid \exists \hat{C}. \hat{D} \leq \hat{C} \wedge (A_L, B_L, \hat{C}) \in \text{dfsync}\}$$

We proceed by case analysis on the structure of A_L; most of the cases are trivial and follow a similar style to the previous proofs. An important case that we will present is when $A_L = \downarrow_L^S A_S$ for some A_S. We will take the case when $B_L = \downarrow_L^S B_S$ for some B_S since the case when $B_L = \downarrow_L^L B'_L$ follows a similar pattern.

$$\begin{aligned}
(\downarrow_L^S A_S, B_L, \hat{D}) &\in \text{dfsync}_{\hat{\Downarrow}}, (\downarrow_L^S A_S, B_L, \hat{C}) \in \text{dfsync} && \text{(this case)}\\
(A_S, B_S, \top) &\in \text{dfsync}, A_S \leq \hat{C} && \text{(by inversion on } D{\downarrow_L^S})\\
(A_S, B_S, \top) &\in \text{dfsync}' && \text{(since dfsync} \subseteq \text{dfsync}')\\
(\downarrow_L^S A_S, \downarrow_L^S C_S, E_S) &\in F(\text{dfsync}') && \text{(by } D{\downarrow_L^S} \text{ with } A_S \leq \hat{D} \text{ by transitivity of } \leq)
\end{aligned}$$

□

The final lemma again deals with the constraint; if we find that the same pair of types is safely synchronizing with respect to two constraints, we can replace it with its meet defined in Lemma 2:

Lemma 13. *If* $\vdash (A_L, B_L, \hat{C})$ *dfsync and* $\vdash (A_L, B_L, \hat{D})$ *dfsync, then* $\vdash (A_L, B_L, \hat{C} \wedge \hat{D})$ *dfsync for some* $A_L, B_L, \hat{C}$, *and* $\hat{D}$.

Proof. First, recall that $\vdash (A_L, B_L, -)$ dfsync requires that $A_L \leq B_L$. We want to show that

$$\text{dfsync}' ::= \text{dfsync} \cup \text{dfsync}_\wedge$$

is F-consistent with

$$\text{dfsync}_\wedge ::= \{(A_L, B_L, \hat{C} \wedge \hat{D}) \mid (A_L, B_L, \hat{C}) \in \text{dfsync} \wedge (A_L, B_L, \hat{D}) \in \text{dfsync}$$

As per usual, we will prove F-consistency of dfsync$'$, that is, dfsync$' \in F(\text{dfsync}')$ by showing that each of the two sets dfsync and dfsync$_\wedge$ are subsets of $F(\text{dfsync}')$.

dfsync $\subseteq F(\text{dfsync}')$ immediately follows from the same argument as in previous lemmas.

We will now consider dfsync$_\wedge \in F(\text{dfsync}')$ by case analysis on the structure of A_L. We can infer the structure of B_L by inversion on the appropriate subtyping rule. For ease of presentation, let $\hat{E} = \hat{C} \wedge \hat{D}$; we will expand $\hat{E}$ whenever necessary.

Case 1. $A_L = \uparrow_L^L A'_L$; then $B_L = \uparrow_L^L B'_L$ with $A'_L \leq B'_L$.

$$\begin{aligned}
&(\uparrow_L^L A'_L, \uparrow_L^L B'_L, \hat{E}) \in \text{dfsync}_\wedge, (\uparrow_L^L A'_L, \uparrow_L^L B'_L, \hat{C}) \in \text{dfsync}, (\uparrow_L^L A'_L, \uparrow_L^L B'_L, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
&\quad (A'_L, B'_L, \hat{C}) \in \text{dfsync}, (A'_L, B'_L, \hat{D}) \in \text{dfsync} && \text{(by inversion on } D\uparrow_L^L)\\
&\quad (A'_L, B'_L, \hat{E}) \in \text{dfsync}_\wedge && \text{(by definition of dfsync}_\wedge)\\
&\quad (A'_L, B'_L, \hat{E}) \in \text{dfsync}' && \text{(since dfsync}_\wedge \subseteq \text{dfsync}')\\
&(\uparrow_L^L A'_L, \uparrow_L^L B'_L, \hat{E}) \in F(\text{dfsync}') && \text{(by } D\uparrow_L^L)
\end{aligned}$$

$\downarrow_L^L, \otimes$, and $\multimap$ follow a similar pattern of appealing to the continuation types.

Case 2. $A_L = \oplus\{\overline{l{:}A_L}\}$; then $B_L = \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}$; with $A_{i_L} \leq B_{i_L}\ \forall i \in \overline{l}$.

$$\begin{aligned}
&(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \hat{E}) \in \text{dfsync}_\wedge\\
&(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \hat{C}) \in \text{dfsync}, (\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
&\qquad (\forall i \in \overline{l})\ (A_{i_L}, B_{i_L}, \hat{C}) \in \text{dfsync}, (A_{i_L}, B_{i_L}, \hat{D}) \in \text{dfsync}, && \text{(by inversion on } D\oplus)\\
&\qquad (\forall i \in \overline{l})\ (A_{i_L}, B_{i_L}, \hat{E}) \in \text{dfsync}_\wedge && \text{(by definition of dfsync}_\wedge)\\
&\qquad (\forall i \in \overline{l})\ (A_{i_L}, B_{i_L}, \hat{E}) \in \text{dfsync}' && \text{(since dfsync}_\wedge \subseteq \text{dfsync}')\\
&(\oplus\{\overline{l{:}A_L}\}, \oplus\{\overline{l{:}B_L}, \overline{m{:}B_L}\}, \hat{E}) \in F(\text{dfsync}') && \text{(by } D\oplus)
\end{aligned}$$

$D\&$ follows a similar pattern.

Case 3. $A_L = \downarrow_L^S A_S$; then there are two subcases for the structure of B_L. We shall take the case when $B_L = \downarrow_L^S B_S$ with $A_S \leq B_S$, but the other case, when $B_L = \downarrow_L^L B'_L$ follows a similar pattern.
At this point we realize what $\hat{E}$ has to be – either $\hat{E} = \bot$, in which case we want to derive a contradiction for this case (the $\bot$ constraint requires that there be no releases) or $\hat{E} = E_S$ meaning $\hat{E}$ is a non-trivial meet.
Subcase 1. $\hat{E} = \bot$.

$$\begin{aligned}
&(\downarrow_L^S A_S, \downarrow_L^S B_S, \bot) \in \text{dfsync}_\wedge, (\downarrow_L^S A_S, \downarrow_L^S B_S, \hat{C}) \in \text{dfsync}, (\downarrow_L^S A_S, \downarrow_L^S B_S, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
&\quad (A_S, B_S, \top) \in \text{dfsync}, A_S \leq \hat{C} && \text{(by inversion on } D\downarrow_L^S)\\
&\quad (A_S, B_S, \top) \in \text{dfsync}, A_S \leq \hat{D} && \text{(by inversion on } D\downarrow_L^S)\\
&\quad \text{Contradiction} && \text{(since } A_S \text{ is a lower bound of } \hat{C} \wedge \hat{D} \text{ but } A_S \text{ is strictly greater than } \bot)
\end{aligned}$$

Subcase 2. $\hat{E} = E_S$ for some E_S.

$$\begin{aligned}
&(\downarrow_L^S A_S, \downarrow_L^S B_S, E_S) \in \text{dfsync}_\wedge, (\downarrow_L^S A_S, \downarrow_L^S B_S, \hat{C}) \in \text{dfsync}, (\downarrow_L^S A_S, \downarrow_L^S B_S, \hat{D}) \in \text{dfsync} && \text{(this case)}\\
&\quad (A_S, B_S, \top) \in \text{dfsync}, A_S \leq \hat{C}, A_S \leq \hat{D} && \text{(by inversion on } D\downarrow_L^S)\\
&\quad (A_S, B_S, \top) \in \text{dfsync}' && \text{(since dfsync} \subseteq \text{dfsync}')\\
&(\downarrow_L^S A_S, \downarrow_L^S C_S, E_S) \in F(\text{dfsync}')\\
&\text{(by } D\downarrow_L^S \text{ with } A_S \leq E_S \text{ because } A_S \text{ is a lower bound of } \hat{C} \text{ and } \hat{D} \text{ and } E_S \text{ is the greatest lower bound)}
\end{aligned}$$

Unlike the previous lemmas, we require A_L to be linear, so we do not need to consider $\uparrow_L^S$. The case when $A = B = 1$ is trivial. □

D Statics

D.1 Rules

$$\frac{B_L \leq A_L}{\Gamma; y_L{:}B_L \vdash \text{fwd } x_L\ y_L :: (x_L{:}A_L)}\ ID_L \qquad \frac{\hat{B} \leq A_S}{\Gamma, y_S{:}\hat{B} \vdash \text{fwd } x_S\ y_S :: (x_S{:}A_S)}\ ID_S \qquad \frac{\hat{B} \leq A_L}{\Gamma, y_S{:}\hat{B}; \cdot \vdash \text{fwd } x_L\ y_S :: (x_L{:}A_L)}\ ID_{LS}$$

$$\frac{\begin{matrix}\overline{v_S{:}\hat{D} \in \Gamma} \\ \overline{w_S{:}\hat{E} \in \Gamma}\end{matrix} \quad \begin{matrix}\overline{B_L} \leq \overline{B'_L} \\ \overline{\hat{D}} \leq \overline{D'_L} \\ \overline{\hat{E}} \leq \overline{E'_S}\end{matrix} \quad \big(x'_L{:}(A_L \leq A'_L) \leftarrow X_L \leftarrow \overline{y'_L{:}B'_L}, \overline{v'_L{:}D'_L}, \overline{w'_S{:}E'_S} = P\big) \in \Sigma \quad \Gamma; \Delta, x_L{:}A'_L \vdash Q :: (z_L{:}C_L)}{\Gamma; \Delta, \overline{y_L{:}B_L} \vdash x_L \leftarrow X_L \leftarrow \overline{y_L}, \overline{v_S}, \overline{w_S}; Q :: (z_L{:}C_L)}\ SP_{LL}$$

$$\frac{\begin{matrix}\overline{y_S{:}\hat{B} \in \Gamma} \\ \overline{\hat{B}} \leq \overline{B'_S}\end{matrix} \quad \big(x'_S{:}(A_S \leq A'_S) \leftarrow X_S \leftarrow \overline{y'_S{:}B'_S} = P\big) \in \Sigma \quad \Gamma, x_S{:}A'_S; \Delta \vdash Q :: (z_L{:}C_L)}{\Gamma; \Delta \vdash x_S \leftarrow X_S \leftarrow \overline{y_S}; Q :: (z_L{:}C_L)}\ SP_{LS}$$

$$\frac{\begin{matrix}\overline{y_S{:}\hat{B} \in \Gamma} \\ \overline{\hat{B}} \leq \overline{B'_S}\end{matrix} \quad \big(x'_S{:}(A_S \leq A'_S) \leftarrow X_S \leftarrow \overline{y'_S{:}B'_S} = P\big) \in \Sigma \quad \Gamma, x_S{:}A'_S \vdash Q :: (z_S{:}C_S)}{\Gamma \vdash x_S \leftarrow X_S \leftarrow \overline{y_S}; Q :: (z_S{:}C_S)}\ SP_{SS}$$

$$\frac{\hat{A} \leq {\uparrow}^S_L A_L \quad \Gamma, x_S{:}\hat{A}; \Delta, x_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma, x_S{:}\hat{A}; \Delta \vdash x_L \leftarrow \text{acq}_S\ x_S; P :: (z_L{:}C_L)}\ {\uparrow}^S_L L \qquad \frac{\Gamma; \cdot \vdash P :: (x_L{:}A_L)}{\Gamma \vdash x_L \leftarrow \text{acc}_S\ x_S; P :: (x_S{:}{\uparrow}^S_L A_L)}\ {\uparrow}^S_L R$$

$$\frac{\Gamma, x_S{:}A_S; \Delta \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}{\downarrow}^S_L A_S \vdash x_S \leftarrow \text{rel}_S\ x_S; P :: (z_L{:}C_L)}\ {\downarrow}^S_L L \qquad \frac{\Gamma \vdash P :: (x_S{:}A_S)}{\Gamma; \cdot \vdash x_S \leftarrow \text{det}_S\ x_S; P :: (x_L{:}{\downarrow}^S_L A_S)}\ {\downarrow}^S_L R$$

$$\frac{\Gamma; \Delta, y_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}{\uparrow}^L_L A_L \vdash y_L \leftarrow \text{acq}_L\ x_L; P :: (z_L{:}C_L)}\ {\uparrow}^L_L L \qquad \frac{\Gamma; \Delta \vdash P :: (x_L{:}A_L)}{\Gamma; \Delta \vdash y_L \leftarrow \text{acc}_L\ x_L; P :: (x_L{:}{\uparrow}^L_L A_L)}\ {\uparrow}^L_L R$$

$$\frac{\Gamma; \Delta, y_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}{\downarrow}^L_L A_L \vdash y_L \leftarrow \text{rel}_L\ x_L; P :: (z_L{:}C_L)}\ {\downarrow}^L_L L \qquad \frac{\Gamma; \Delta \vdash P :: (y_L{:}A_L)}{\Gamma; \Delta \vdash y_L \leftarrow \text{det}_L\ x_L; P :: (x_L{:}{\downarrow}^L_L A_L)}\ {\downarrow}^L_L R$$

$$\frac{\Gamma; \Delta \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}1 \vdash \text{wait } x_L; P :: (z_L{:}C_L)}\ 1L \qquad \frac{}{\Gamma; \cdot \vdash \text{close } x_L :: (x_L{:}1)}\ 1R$$

$$\frac{\Gamma; \Delta, x_L{:}B_L, y_L{:}A_L \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}A_L \otimes B_L \vdash y_L \leftarrow \text{recv } x_L; P :: (z_L{:}C_L)}\ {\otimes}L \qquad \frac{A'_L \leq A_L \quad \Gamma; \Delta \vdash P :: (x_L{:}B_L)}{\Gamma; \Delta, y_L{:}A'_L \vdash \text{send } x_L\ y_L; P :: (x_L{:}A_L \otimes B_L)}\ {\otimes}R$$

$$\frac{A'_L \leq A_L \quad \Gamma; \Delta, x_L{:}B \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}A_L \multimap B_L, y_L{:}A'_L \vdash \text{send } x_L\ y_L; P :: (z_L{:}C_L)}\ {\multimap}L \qquad \frac{\Gamma; \Delta, y_L{:}A_L \vdash P :: (x_L{:}B_L)}{\Gamma; \Delta \vdash y_L \leftarrow \text{recv } x_L; P :: (x_L{:}A_L \multimap B_L)}\ {\multimap}R$$

$$\frac{\hat{A} \leq A_L \quad \Gamma, y_S{:}\hat{A}; \Delta, x_L{:}B \vdash P :: (z_L{:}C_L)}{\Gamma, y_S{:}\hat{A}; \Delta, x_L{:}A_L \multimap B_L \vdash \text{send } x_L \ y_S; P :: (z_L{:}C_L)} \multimap L_S \qquad \frac{\hat{A} \leq A_L \quad \Gamma, y_S{:}\hat{A}; \Delta \vdash P :: (x_L{:}B_L)}{\Gamma, y_S{:}\hat{A}; \Delta \vdash \text{send } x_L \ y_S; P :: (x_L{:}A_L \otimes B_L)} \otimes R_S$$

$$\frac{\forall i \in \bar{l} \quad \Gamma; \Delta, x_L{:}A_{i_L} \vdash P_i :: (c_L{:}Z_L)}{\Gamma; \Delta, x_L{:} \oplus \{\overline{l{:}A_L}\} \vdash \text{case } x_L \text{ of } \{\overline{l \Rightarrow P}\} :: (c_L{:}Z_L)} \oplus L \qquad \frac{i \in \bar{l} \quad \Gamma; \Delta \vdash P :: (x_L{:}A_{i_L})}{\Gamma; \Delta \vdash x.i; P :: (x_L{:} \oplus \{\overline{l{:}A_L}\})} \oplus R$$

$$\frac{i \in \bar{l} \quad \Gamma; \Delta, x_L{:}A_{i_L} \vdash P :: (z_L{:}C_L)}{\Gamma; \Delta, x_L{:}\&\{\overline{l{:}A_L}\} \vdash x.i; P :: (z_L{:}C_L)} \& L \qquad \frac{\forall i \in \bar{l} \quad \Gamma; \Delta \vdash P_i :: (x_L{:}A_{i_L})}{\Gamma; \Delta \vdash \text{case } x_L \text{ of } \{\overline{l \Rightarrow P}\} :: (x_L{:}\&\{\overline{l{:}A_L}\})} \& R$$

E Dynamics

$$\text{proc}(a_L, \text{fwd } a_L\ b_L), \Psi_b \to \Psi_b \quad (b_L := a_L, b_S := a_S) \tag{D-FWDLL}$$

$$\text{proc}(a_L, \text{fwd } a_L\ b_S) \to \text{connect}(a_L, b_S) \tag{D-FWDLS}$$

$$\text{proc}(a_S, \text{fwd } a_S\ b_S) \to \text{unavail}(a_S) \quad (b_S := a_S) \tag{D-FWDSS}$$

$$\begin{array}{c}\text{proc}(a_L, x_L \leftarrow X_L \leftarrow \overline{b_L}, \overline{d_S}, \overline{e_S}; Q)\\ !\text{def}((x'_L{:}(A_L \leq A'_L)) \leftarrow X_L \leftarrow \overline{y'_L{:}B'_L}, \overline{v'_L{:}D'_L}, \overline{w'_S{:}E'_S}) = P)\end{array} \to \begin{array}{c}\text{proc}(a_L, [c_L/x_L]Q), \text{proc}(b_L, [c_L/x'_L, \overline{b_L}/\overline{y'_L}, \overline{d'_L}/\overline{v'_L}, \overline{e_S}/\overline{w'_S}]P)\\ \overline{\text{connect}(d'_L, d_S), \text{unavail}(d'_S)} \quad (\overline{d'}, c \text{ fresh})\end{array} \tag{D-SPAWNLL}$$

$$\begin{array}{c}\text{proc}(a_L, x_S \leftarrow X_S \leftarrow \overline{b_S}; Q)\\ !\text{def}((x'_S{:}(A_S \leq A'_S)) \leftarrow X_S \leftarrow \overline{y'_S{:}B'_S}) = P)\end{array} \to \text{proc}(a_L, [c_S/x_S]Q), \text{proc}(c_S, [c_S/x'_S, \overline{b_S}/\overline{y'_S}]P) \quad (c \text{ fresh}) \tag{D-SPAWNLS}$$

$$\begin{array}{c}\text{proc}(a_S, x_S \leftarrow X_S \leftarrow \overline{b_S}; Q)\\ !\text{def}((x'_S{:}(A_S \leq A'_S)) \leftarrow X_S \leftarrow \overline{y'_S{:}B'_S}) = P)\end{array} \to \text{proc}(a_S, [c_S/x_S]Q), \text{proc}(c_S, [c_S/x'_S, \overline{b_S}/\overline{y'_S}]P) \quad (c \text{ fresh}) \tag{D-SPAWNSS}$$

$$\text{proc}(a_L, x_L \leftarrow \text{acq}_S\ b_S; P), \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q) \to \text{proc}(a_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L]Q), \text{unavail}(b_S) \tag{D-$\uparrow_L^S$}$$

$$\text{proc}(a_L, x_S \leftarrow \text{rel}_S\ b_S; P), \text{proc}(b_L, x_S \leftarrow \text{det}_S\ b_S; Q), \text{unavail}(b_S) \to \text{proc}(a_L, [b_S/x_S]P), \text{proc}(b_S, [b_S/x_S Q) \tag{D-$\downarrow_L^S$}$$

$$\text{proc}(a_L, x_L \leftarrow \text{acq}_L\ b_L; P), \text{proc}(b_L, x_L \leftarrow \text{acc}_L\ b_L; Q) \to \text{proc}(a_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L]Q) \tag{D-$\uparrow_L^L$}$$

$$\text{proc}(a_L, x_L \leftarrow \text{rel}_L\ b_L; P), \text{proc}(b_L, x_L \leftarrow \text{det}_L\ b_L; Q) \to \text{proc}(a_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L Q) \tag{D-$\downarrow_L^L$}$$

$$\begin{array}{c}\text{proc}(a_L, x_L \leftarrow \text{acq}_L\ b_L; P), \text{connect}(b_L, c_S)\\ \text{proc}(c_S, x_L \leftarrow \text{acc}_S\ c_S; Q)\end{array} \to \begin{array}{c}\text{proc}(a_L, [c_L/x_L]P), \text{proc}(c_L, [c_L/x_L]Q)\\ \text{unavail}(c_S)\end{array} \tag{D-$\uparrow_L^S$2}$$

$$\begin{array}{c}\text{proc}(a_L, x_L \leftarrow \text{rel}_L\ c_L; P), \text{proc}(c_L, x_S \leftarrow \text{det}_S\ c_S; Q),\\ \text{unavail}(c_S)\end{array} \to \begin{array}{c}\text{proc}(a_L, [b_L/x_L]P), \text{connect}(b_L, c_S),\\ \text{unavail}(b_S)\end{array} \tag{D-$\downarrow_L^S$2}$$

$$\text{proc}(c_S, [c_S/x_S]Q) \quad (b \text{ fresh})$$

$$\text{proc}(a_L, \text{wait } b_L; P), \text{proc}(b_L, \text{close } b_L) \to \text{proc}(a_L, P) \tag{D-1}$$

$$\text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_L; Q), \Psi_c \to \text{proc}(a_L, [c_L/y_L]P), \text{proc}(b_L, Q), \Psi_c \tag{D-$\otimes$}$$

$$\text{proc}(a_L, \text{send } b_L\ c_L; P), \text{proc}(b_L, y_L \leftarrow \text{recv } b_L; Q), \Psi_c \to \text{proc}(a_L, P), \text{proc}(b_L, [c_L/y_L]Q), \Psi_c \tag{D-$\multimap$}$$

$$\text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_S; Q) \to \text{proc}(a_L, [d_L/y_L]P), \text{proc}(b_L, Q), \text{connect}(d_L, c_S), \text{unavail}(d_S) \quad (d \text{ fresh}) \tag{D-$\otimes$2}$$

$$\text{proc}(a_L, \text{send } b_L\ c_S; P), \text{proc}(b_L, y_L \leftarrow \text{recv } b_L; Q) \to \text{proc}(a_L, P)\text{proc}(b_L, [d_L/y_L]Q), \text{connect}(d_L, c_S), \text{unavail}(d_S) \quad (d \text{ fresh}) \tag{D-$\multimap$2}$$

$$\text{proc}(a_L, \text{case } b_L \text{ of } \{\overline{l \Rightarrow P}, \overline{m \Rightarrow P}\}), \text{proc}(b_L, b.i; Q) \to \text{proc}(a_L, P_i), \text{proc}(b_L, Q) \quad (i \in \overline{l}) \tag{D-$\oplus$}$$

$$\text{proc}(a_L, b.i; P), \text{proc}(b_L, \text{case } b_L \text{ of } \{\overline{l \Rightarrow Q}, \overline{m \Rightarrow Q}\}) \to \text{proc}(a_L, P), \text{proc}(b_L, Q_i) \quad (i \in \overline{l}) \tag{D-&}$$

F Preservation

Theorem 1. *If* $\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)$ *and* $\Lambda; \Theta \to \Lambda'; \Theta'$, *then* $\Gamma' \models \Lambda'; \Theta' :: (\Gamma'; \Delta)$ *for some* Λ', Θ', *and* Γ' *such that* $\Gamma' \preceq \Gamma$.

Proof. By induction on the dynamics to construct a well-formed and well-typed configuration starting with $\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)$.

Notation Many of the proof cases involve transitions between linear process terms (either proc or connect). When reasoning with these transitions, we adopt the notation that $\Psi_a \to \Psi'_a$ that is, Ψ_a represents the process term offering a before the transition and Ψ'_a represents the process term offering a after the transition.

Case 1. D-FWDLS

$$\text{proc}(a_L, \text{fwd } a_L\ b_S) \to \text{connect}(a_L, b_S)$$

where $\Psi_a = \text{proc}(a_L, \text{fwd } a_L\ b_S)$ and $\Psi'_a = \text{connect}(a_L, b_S)$ (for the remaining cases, these metavariable assignments are implicit). Let $\Theta = \Theta_1, \Psi_a, \Theta_2$. Then by Lemma 9, $\Lambda = \text{unavail}(a_S), \Lambda_1$.

$$\begin{array}{lr}
\Gamma \models \Lambda; \Theta_1, \Psi_a, \Theta_2 :: (\Gamma; \Delta) & \text{(assumption)} \\
\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2 :: (\Delta) & \text{(by inversion on } \Omega) \\
\Gamma \models \text{proc}(a_L, \text{fwd } a_L\ b_S), \Theta_2 :: (a_L{:}A_L, \Delta_p) & \text{(by Lemma 4 and expanding } \Psi_a) \\
\Gamma \models \Theta_2 :: (\Delta_p) \quad \Gamma; \cdot \vdash \text{fwd } a_L\ b_S :: (a_L{:}A'_L) & \text{(by inversion on } \Theta 3) \\
b_S{:}\hat{B} \in \Gamma \quad \hat{B} \leq A'_L & \text{(by inversion on } ID_{LS}) \\
\hat{B} \leq A_L & \text{(by transitivity of } \leq) \\
\Gamma \models \text{connect}(a_L, b_S), \Theta_2 :: (a : A_L, \Delta_p) & \text{(by } \Theta 2) \\
\Gamma \models \Theta_1, \Psi'_a, \Theta_2 :: (\Delta) & \text{(by Lemma 5)} \\
\Gamma \models \Lambda; \Theta_1, \Psi'_a, \Theta_2 :: (\Gamma; \Delta) & \text{(by } \Omega)
\end{array}$$

The well-formedness conditions are maintained because only $\Psi_a \in \Theta$ was replaced by Ψ'_a.

Case 2. D-&

$$\text{proc}(a_L, b.i; P), \text{proc}(b_L, \text{case } b_L \text{ of } \{\overline{l \Rightarrow Q}, \overline{m \Rightarrow Q}\}) \to \text{proc}(a_L, P), \text{proc}(b_L, Q_i) \quad (i \in \overline{l})$$

Then $\Theta = \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3$

$\Gamma \models \Lambda; \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Gamma; \Delta)$ (assumption)

$\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Delta)$ (by inversion on Ω)

$\Gamma \models \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (a_L{:}A_L, \Delta_r)$ (by Lemma 4)

$\Gamma \models \Psi_a, \Psi_b, \Theta_r :: (a_L{:}A_L, \Delta_r)$ (by Lemma 6 and $\Theta_r = \Theta_2, \Theta_3$)

$\Gamma \models \Psi_b, \Theta_r :: (b_L{:}\&\{\overline{l{:}B_L}\}, \Delta_a, \Delta_r) \quad \Gamma; \Delta_a \vdash b.i; P :: (a_L{:}A'_L) \quad a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})$ dfsync
(by inversion on $\Theta 3$)

$\Gamma \models \Theta_r :: (\Delta_a, \Delta_b, \Delta_r) \quad \Gamma; \Delta_b \vdash \text{case } b_L \text{ of } \{\overline{l \Rightarrow Q}, \overline{m \Rightarrow Q}\} :: (b_L{:}\&\{\overline{l{:}B'_L}, \overline{m{:}B'_L}\})$

$b_S{:}\hat{B} \in \Gamma \quad \vdash (\&\{\overline{l{:}B'_L}, \overline{m{:}B'_L}\}, \&\{\overline{l{:}B_L}\}, \hat{B})$ dfsync (by inversion on $\Theta 3$)

$\Gamma; \Delta_b \vdash Q_i :: (b_L{:}B'_{i_L})$ (inversion on $\& R$)

$B'_{i_L} \leq B_{i_L} \quad \vdash (B'_{i_L}, B_{i_L}, \hat{B})$ dfsync (by inversion on $\leq_\&$ and E& respectively)

$\Gamma \models \Psi'_b, \Theta_r :: (b_L{:}B_{i_L}, \Delta_r, \Delta_a)$ (by $\Theta 3$)

$\Gamma; \Delta_a, b_L{:}B_{i_L} \vdash P :: (a_L{:}A'_L)$ (inversion on $\& L$)

$\Gamma \models \Psi'_a, \Psi'_b, \Theta_r :: (a_L{:}A_L, \Delta_r)$ (by $\Theta 3$)

$\Gamma \models \Theta_1, \Psi'_a, \Psi'_b, \Theta_r :: (\Delta)$ (by Lemma 5)

$\Gamma \models \Lambda; \Theta_1, \Psi'_a, \Psi'_b, \Theta_r :: (\Gamma; \Delta)$ (by Ω)

The well-formedness conditions are maintained because Ψ_a and Ψ_b were replaced by Ψ'_a and Ψ'_b respectively in Θ.

The proof of D-$\oplus$ is similar to D-&.

Case 3. D-$\otimes$

$$\text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_L; Q), \Psi_c \rightarrow \text{proc}(a_L, [c_L/y_L]P), \text{proc}(b_L, Q), \Psi_c$$

Then $\Theta = \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3, \Psi_c, \Theta_4$.

$\Gamma \models \Lambda; \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3, \Psi_c, \Theta_4 :: (\Gamma; \Delta)$ (assumption)

$\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3, \Psi_c, \Theta_4 :: (\Delta)$ (by inversion on Ω)

$\Gamma \models \Psi_a, \Theta_2, \Psi_b, \Theta_3, \Psi_c, \Theta_4 :: (a_L{:}A_L, \Delta_r)$ (by Lemma 4)

$\Gamma \models \Psi_a, \Psi_b, \Psi_c, \Theta_r :: (a_L{:}A_L, \Delta_r)$ (by Lemma 6 and $\Theta_r = \Theta_2, \Theta_3, \Theta_4$)

$\Gamma \models \Psi_b, \Psi_c, \Theta_r :: (b_L{:}C_L^a \otimes B_L, \Delta_a, \Delta_r) \quad \Gamma; \Delta_a, b_L{:}B_L \vdash y_L \leftarrow \text{recv } b_L; P :: (a_L{:}A'_L)$

$a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync}$ (by inversion on $\Theta 3$)

$\Gamma; \Delta_a, b_L{:}B_L, c_L{:}C_L^a \vdash [c_L/y_L]P :: (a_L{:}A'_L)$ (by inversion on $\otimes L$ and α equivalance)

$\Gamma \models \Psi_c, \Theta_r :: (c_L{:}C_L, \Delta_a, \Delta_b, \Delta_r) \quad \Gamma; \Delta_b, c_L{:}C_L \vdash \text{send } b_L\ c_L; Q :: (b_L{:}C_L^b \otimes B'_L)$

$b_S{:}\hat{B} \in \Gamma \quad \vdash (C_L^b \otimes B'_L, C_L^a \otimes B_L, \hat{B})\ \text{dfsync}$ (by inversion on $\Theta 3$)

$\Gamma; \Delta_b \vdash Q :: (b_L{:}B'_L) \quad C_L \leq C_L^b$ (by inversion on $\otimes R$)

$C_L^b \leq C_L^a \quad B'_L \leq B_L \quad \vdash (B'_L, B_L, \hat{B})\ \text{dfsync}$ (by inversion on $\leq_\otimes$ and $E\otimes$ respectively)

$\Gamma \models \Psi_c, \Theta_r :: (c_L{:}C_L^a, \Delta_a, \Delta_b, \Delta_r)$ (by Lemma 7 since $C_L \leq C_L^b \leq C_L^a$.)

$\Gamma \models \Psi'_b, \Psi_c, \Theta_r :: (b_L{:}B_L, c_L{:}C_L^a, \Delta_a, \Delta_r)$ (by $\Theta 3$)

$\Gamma \models \Psi'_a, \Psi'_b, \Psi_c, \Theta_r :: (a_L{:}A_L, \Delta_r)$ (by $\Theta 3$)

$\Gamma \models \Theta_1, \Psi'_a, \Psi'_b, \Psi_c, \Theta_r :: (\Delta)$ (by Lemma 5)

$\Gamma \models \Lambda; \Theta_1, \Psi'_a, \Psi'_b, \Psi_c, \Theta_r :: (\Gamma; \Delta)$ (by Ω)

The well-formedness conditions are maintained because Ψ_a and Ψ_b were replaced by Ψ'_a and Ψ'_b respectively in Θ.

Case 4. D-$\otimes$2

$$\begin{aligned} & \text{proc}(a_L, y_L \leftarrow \text{recv } b_L; P), \text{proc}(b_L, \text{send } b_L\ c_S; Q) \\ \rightarrow\ & \text{proc}(a_L, [d_L/y_L]P), \text{proc}(b_L, Q), \text{connect}(d_L, c_S), \text{unavail}(d_S) \quad (d\ \text{ fresh}) \end{aligned}$$

Then $\Theta = \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3$.

$$
\begin{array}{ll}
\Gamma \models \Lambda; \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Gamma; \Delta) & \text{(assumption)} \\
\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Delta) & \text{(by inversion on } \Omega) \\
\Gamma \models \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (a_L{:}A_L, \Delta_r) & \text{(by Lemma 4)} \\
\Gamma \models \Psi_a, \Psi_b, \Theta_r :: (a_L{:}A_L, \Delta_r) & \text{(by Lemma 6 and } \Theta_r = \Theta_2, \Theta_3) \\
\Gamma \models \Psi_b, \Theta_r :: (b_L{:}C^a_L \otimes B_L, \Delta_a, \Delta_r) \quad \Gamma; \Delta_a, b_L{:}B_L \vdash y_L \leftarrow \text{recv } b_L; P :: (a_L{:}A'_L) & \\
a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A}) \text{ dfsync} & \text{(by inversion on } \Theta 3) \\
\Gamma; \Delta_a, b_L{:}B_L, d_L{:}C^a_L \vdash [d_L/y_L]P :: (a_L{:}A'_L) & \text{(by inversion on } \otimes L \text{ and } \alpha \text{ equivalance)} \\
\Gamma \models \Theta_r :: (\Delta_a, \Delta_b, \Delta_r) \quad \Gamma; \Delta_b, c_L{:}C_L \vdash \text{send } b_L\ c_S; Q :: (b_L{:}C^b_L \otimes B'_L) & \\
b_S{:}\hat{B} \in \Gamma \quad \vdash (C^b_L \otimes B'_L, C^a_L \otimes B_L, \hat{B}) \text{ dfsync} & \text{(by inversion on } \Theta 3) \\
\Gamma; \Delta_b \vdash Q :: (b_L{:}B'_L) \quad \hat{C} \leq C^b_L & \text{(by inversion on } \otimes R_S) \\
\Gamma \models \text{connect}(d_L, c_S), \Theta_r :: (d : C^a_L, \Delta_a, \Delta_b, \Delta_r) & \\
\Gamma \models \Psi'_b, \Psi_d, \Theta_r :: (b_L{:}B_L, d_L{:}C^a_L, \Delta_a, \Delta_r) & \text{(by } \Theta 3 \text{ where } \Psi_d = \text{connect}(d_L, c_S)) \\
\Gamma \models \Psi'_a, \Psi'_b, \Psi_d, \Theta_r :: (a_L{:}A_L, \Delta_r) & \text{(by } \Theta 3) \\
\Gamma \models \Theta_1, \Psi'_a, \Psi'_b, \Psi_d, \Theta_r :: (\Delta) & \text{(by Lemma 5)} \\
\Gamma' \models \Theta_1, \Psi'_a, \Psi'_b, \Psi_d, \Theta_r :: (\Delta) & \text{(by Lemma 8 with } \Gamma' = \Gamma, d_S{:}\bot) \\
\Gamma' \models \Lambda :: (\Gamma) & \text{(by Lemma 8)} \\
\Gamma' \models \text{unavail}(d_S) :: (d_S{:}\bot) & \text{(by } \Lambda 4) \\
\Gamma' \models \Lambda, \text{unavail}(d_S) :: (\Gamma') & \text{(by } \Lambda 2) \\
\Gamma' \models \Lambda, \text{unavail}(d_S); \Theta_1, \Psi'_a, \Psi'_b, \Psi_d, \Theta_r :: (\Gamma'; \Delta) & \text{(by } \Omega)
\end{array}
$$

The well-formedness conditions are maintained because Ψ_a and Ψ_b were replaced by Ψ'_a and Ψ'_b respectively in Θ and a Ψ_d was added in Θ where d is fresh along with a corresponding unavail(d_S) in $\Lambda' = \Lambda, \text{unavail}(d_S)$.

The proofs of D-$\multimap$ and D-$\multimap$2 are similar to D-$\otimes$ and D-$\otimes$2 respectively.

We will now present some of the harder cases:

Case 5. D-FWDLL

$$\text{proc}(a_L, \text{fwd } a_L\ b_L), \Psi_b \rightarrow \Psi_b \quad (a_L := b_L, a_S := b_S)$$

Then $\Theta = \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3$ and $\Lambda = \text{unavail}(a_S), \text{unavail}(b_S), \Lambda_1$ by Lemma 9.

$\Gamma \models \Lambda; \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Gamma; \Delta)$ (assumption)

$\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Delta)$ (by inversion on Ω)

$\Gamma \models \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (a_L{:}A_L, \Delta_r)$ (by Lemma 4)

$\Gamma \models \Psi_a, \Psi_b, \Theta_r :: (a_L{:}A_L, \Delta_r)$ (by Lemma 6 and $\Theta_r = \Theta_2, \Theta_3$)

$\Gamma \models \Psi_b, \Theta_r :: (b_L{:}b_L{:}B_L, \Delta_r) \quad \Gamma; b_L{:}B_L \vdash \text{fwd } a_L\ b_L :: (a_L{:}A'_L) \quad a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A})\ \text{dfsync}$
(by inversion on $\Theta3$)

$B_L \leq A'_L$ (by inversion on ID_L)

At this point we need to case on the structure of Ψ_b. In both cases we will show that $\Gamma' \models \Psi'_a, \Theta_r :: (a_L{:}A_L, \Delta_r)$ for some $\Gamma' \preceq \Gamma$ and Ψ'_a being directly defined from Ψ_b.

Subcase 1. $\Psi_b = \text{connect}(b_L, c_S)$ for some c_S.

$\Gamma \models \text{connect}(b_L, c_S), \Theta_r :: (b : B_L, \Delta_r) \quad c_S{:}\hat{C} \in \Gamma \quad \hat{C} \leq B_L$ (by inversion on $\Theta2$)

$\hat{C} \leq A_L$ (by transitivity of $\leq$)

$\Gamma \models \text{connect}(b_L, c_S), \Theta_r :: (b : A_L, \Delta_r)$ (by $\Theta2$)

$\Gamma \models \text{connect}(a_L, c_S), \Theta_r :: (a : A_L, \Delta_r)$ (from renaming)

Subcase 2. $\Psi_b = \text{proc}(b_L, P)$ for some process term P.

$\Gamma \models \Theta_r :: (\Delta_b, \Delta_r) \quad \Gamma; \Delta_b \vdash P :: (b_L{:}B'_L) \quad b_S{:}\hat{B} \in \Gamma \quad \vdash (B'_L, B_L, \hat{B})\ \text{dfsync}$ (by inversion on $\Theta3$)

$B'_L \leq B_L \leq A'_L \leq A_L$

$\vdash (B'_L, A_L, \hat{B})\ \text{dfsync} \quad \vdash (B'_L, A_L, \hat{A})\ \text{dfsync}$ (by Lemma 10 and Lemma 11 respectively)

$\vdash (B'_L, A_L, \hat{B} \wedge \hat{A})\ \text{dfsync}$ (by Lemma 13)

$\Gamma' \models \Theta_r :: (\Delta_b, \Delta_r)$ (by Lemma 8 with $\Gamma' = [a_S{:}\hat{B} \wedge \hat{A}/a_S{:}\hat{A}]\Gamma$)

$\Gamma'; \Delta_b \vdash P :: (b_L{:}B'_L)$ (by Lemma 3)

$\Gamma'; \Delta_b \vdash [a_L/b_L, a_S/b_S]P :: (a_L{:}B'_L)$
(by α equivalence for a_L/b_L and a combination of α equivalence and Lemma 8 for a_S/b_S)

$\Gamma' \models \text{proc}(a_L, [a_L/b_L, a_S/b_S]P), \Theta_r :: (a : A_L, \Delta_r)$ (by $\Theta3$)

We will now continue assuming $\Gamma' \models \Psi'_a, \Theta_r :: (a_L{:}A_L, \Delta_r)$ with $\Gamma' \preceq \Gamma$ and $\Psi'_a = [a_L/b_L, a_S/b_S]\Psi_b$.

For the connect case that did not require a smaller Γ, simply set $\Gamma' = \Gamma$ since $\Gamma' \preceq \Gamma$ by reflexivity.

$$
\begin{array}{ll}
\Gamma' \models \Theta_1, \Psi_a, \Theta_r :: (\Delta) & \text{(by Lemma 8)} \\
\Gamma' \models \Theta_1, \Psi'_a, \Theta_r :: (\Delta) & \text{(by Lemma 5)} \\
\Gamma' \models \Lambda :: (\Gamma) & \text{(by Lemma 5)} \\
\Gamma' \models \text{unavail}(a_S) :: (a_S{:}\bot) & \text{(by } \Lambda 4\text{)} \\
\Gamma' \models \text{unavail}(b_S), \Theta_1 :: (\Gamma'') & \text{(by inversion on } \Lambda 2 \text{ where } \Gamma' = \Gamma'', a_S{:}\bot) \\
\Gamma' \models \Lambda :: (\Gamma') & \text{(by } \Lambda 2\text{)} \\
\Gamma' \models [a_S/b_S]\Lambda :: (\Gamma') & \text{(by } \alpha \text{ equivalence)} \\
\Gamma' \models [a_S/b_S]\Theta_1, \Psi'_a, [a_S/b_S]\Theta_r :: (\Delta) & \text{(by } \alpha \text{ equivalence)} \\
\Gamma' \models \Lambda; [a_S/b_S, a_L/b_L]\Theta_1, \Psi'_a, [a_S/b_S]\Theta_r :: (\Gamma'; \Delta) & \text{(by } \Omega\text{)}
\end{array}
$$

Well-formedness is easily maintained because we only removed something from the linear fragment (it is okay to have dangling unavail terms in the shared fragment).

Case 6. D-$\uparrow_L^S$

$$
\begin{array}{ll}
& \text{proc}(a_L, x_L \leftarrow \text{acq}_S\ b_S; P), \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q) \\
\rightarrow & \text{proc}(a_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L]Q), \text{unavail}(b_S)
\end{array}
$$

Then $\Lambda = \Lambda_b, \Lambda_1$ and $\Theta = \Theta_1, \Psi_a, \Theta_2$ with $\Lambda_b = \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q)$.

We also define $\Psi'_b = \text{proc}(b_L, [b_L/x_L]Q)$.

$$\begin{array}{lr}
\Gamma \models \Lambda_b, \Lambda_1; \Theta_1, \Psi_a, \Theta_2 :: (\Gamma; \Delta) & \text{(assumption)} \\
\Gamma \models \Lambda_b, \Lambda_1 :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2 :: (\Delta) & \text{(by inversion on } \Omega) \\
\Gamma \models \Lambda_b :: (b_S{:}\uparrow_L^S B_L) \quad \Gamma \models \Lambda_1 :: (\Gamma') & \text{(by inversion on } \Lambda 2 \text{ with } \Gamma = b_S{:}\uparrow_L^S B_L, \Gamma') \\
\vdash (\uparrow_L^S B'_L, \uparrow_L^S B_L, \top) \text{ dfsync} \quad \Gamma \vdash x_L \leftarrow \text{acc}_S\ b_S; Q :: (b_S{:}\uparrow_L^S B'_L) & \text{(by inversion on } \Lambda 3) \\
\Gamma; \cdot \vdash [b_L/x_L]Q :: (b_L{:}B'_L) & \text{(by inversion on } \uparrow_L^S R \text{ and } \alpha \text{ equivalence)} \\
\vdash (B'_L, B_L, \uparrow_L^S B'_L) \text{ dfsync} & \text{(by inversion on } D\uparrow_L^S) \\
\Gamma \models \Psi_a, \Theta_2 :: (a_L{:}A_L, \Delta_p) & \text{(by Lemma 4)} \\
\Gamma \models \Theta_2 :: (\Delta_a, \Delta_p) \quad \Gamma; \Delta_a \vdash x_L \leftarrow \text{acq}_S\ b_S :: (a_L{:}A'_L) & \\
a_S{:}\hat{A} \in \Gamma \quad \vdash (A'_L, A_L, \hat{A}) \text{ dfsync} & \text{(by inversion on } \Theta 3) \\
\Gamma; \Delta_a, b_L{:}B^a_L \vdash [b_L/x_L]P :: (a_L{:}A'_L) \quad \uparrow_L^S B_L \leq \uparrow_L^S B^a_L & \text{(by inversion on } \uparrow_L^S L \text{ and } \alpha \text{ equivalence)} \\
\Gamma \models \Psi'_b, \Theta_2 :: (b_L{:}B_L, \Delta_a, \Delta_p) & \text{(by } \Lambda 3) \\
\Gamma \models \Psi'_b, \Theta_2 :: (b_L{:}B^a_L, \Delta_a, \Delta_p) & \text{(by Lemma 7)} \\
\Gamma \models \Psi'_a, \Psi'_b, \Theta_2 :: (a_L{:}A, \Delta_p) & \text{(by } \Theta 3) \\
\Gamma \models \Theta_1, \Psi'_a, \Psi'_b, \Theta_2 :: (\Delta) & \text{(by Lemma 5)} \\
\Gamma \models \text{unavail}(b_S) :: (b_S{:}\uparrow_L^S B_L) & \text{(by } \Lambda 4) \\
\Gamma \models \text{unavail}(b_S), \Lambda_1 :: (\Gamma) & \text{(by } \Lambda 2) \\
\Gamma \models \Lambda; \Theta_1, \Psi'_a, \Psi'_b, \Theta_2 :: (\Gamma; \Delta) & \text{(by } \Omega)
\end{array}$$

Well-formedness is maintained because $\Psi_b \notin \Theta$ and there is a corresponding unavail(b_S) to the newly added Ψ'_b.

Case 7. D-$\uparrow_L^S 2$

$$\begin{array}{rl}
& \text{proc}(a_L, x_L \leftarrow \text{acq}_L\ b_L; P), \text{connect}(b_L, c_S), \text{proc}(c_S, x_L \leftarrow \text{acc}_S\ c_S; Q) \\
\rightarrow & \text{proc}(a_L, [c_L/x_L]P), \text{proc}(c_L, [c_L/x_L]Q), \text{unavail}(c_S)
\end{array}$$

Then $\Lambda = \Lambda_c, \Lambda_1$ and $\Theta = \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3$ with $\Lambda_c = \text{proc}(c_S, x_L \leftarrow \text{acc}_S\ c_S; Q)$.

We also define $\Psi_c' = \text{proc}(c_L, [c_L/x_L]Q)$.

$$\begin{array}{lr}
\Gamma \models \Lambda_c, \Lambda_1; \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Gamma; \Delta) & \text{(assumption)} \\
\Gamma \models \Lambda_c, \Lambda_1 :: (\Gamma) \quad \Gamma \models \Theta_1, \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (\Delta) & \text{(by inversion on } \Omega) \\
\Gamma \models \Lambda_c :: (c_S{:}\uparrow_L^S C_L) \quad \Gamma \models \Lambda_1 :: (\Gamma') & \text{(by inversion on } \Lambda 2 \text{ with } \Gamma = c_S{:}\uparrow_L^S C_L, \Gamma') \\
\vdash (\uparrow_L^S C_L', \uparrow_L^S C_L, \top) \text{ dfsync} \quad \Gamma \vdash x_L \leftarrow \text{acc}_S\ c_S; Q :: (c_S{:}\uparrow_L^S C_L') & \text{(by inversion on } \Lambda 3) \\
\Gamma; \cdot \vdash [c_L/x_L]Q :: (c_L{:}C_L') & \text{(by inversion on } \uparrow_L^S R \text{ and } \alpha \text{ equivalence)} \\
\vdash (C_L', C_L, \uparrow_L^S C_L') \text{ dfsync} & \text{(by inversion on } D\uparrow_L^S) \\
\Gamma \models \Psi_a, \Theta_2, \Psi_b, \Theta_3 :: (a_L{:}A_L, \Delta_p) & \text{(by Lemma 4)} \\
\Gamma \models \Psi_a, \Psi_b, \Theta_r :: (a_L{:}A_L, \Delta_p) & \text{(by Lemma 6 with } \Theta_r = \Theta_2, \Theta_3) \\
\Gamma \models \text{connect}(b_L, c_S), \Theta_2 :: (b_S{:}\uparrow_L^L B_L, \Delta_a, \Delta_p) \quad \Gamma; \Delta_a, b_S{:}\uparrow_L^L B_L \vdash x_L \leftarrow \text{acq}_L\ b_L :: (a_L{:}A_L') & \\
a_S{:}\hat{A} \in \Gamma \quad \vdash (A_L', A_L, \hat{A}) \text{ dfsync} & \text{(by inversion on } \Theta 3) \\
\Gamma \models \Theta_r :: (\Delta_a, \Delta_p) \quad \uparrow_L^S C_L \leq \uparrow_L^L B_L & \text{(by inversion on } \Theta 2) \\
C_L \leq B_L \quad \vdash (C_L', B_L, \uparrow_L^S C_L') \text{ dfsync} & \text{(by inversion on } \leq_{\uparrow_L^S \uparrow_L^L} \text{ and Lemma 10 respectively)} \\
\Gamma \models \Psi_c', \Theta_r :: (c_L{:}C_L, \Delta_a, \Delta_p) & \text{(by } \Lambda 3) \\
\Gamma; \Delta_a, c_L{:}C_L \vdash [c_L/x_L]P :: (a_L{:}A_L') & \text{(by inversion on } \uparrow_L^S L \text{ and } \alpha \text{ equivalence)} \\
\Gamma \models \Psi_a', \Psi_c', \Theta_2 :: (a_L{:}A, \Delta_p) & \text{(by } \Theta 3) \\
\Gamma \models \Theta_1, \Psi_a', \Psi_b', \Theta_2 :: (\Delta) & \text{(by Lemma 5)} \\
\Gamma \models \text{unavail}(c_S) :: (c_S{:}\uparrow_L^S C_L) & \text{(by } \Lambda 4) \\
\Gamma \models \text{unavail}(c_S), \Lambda_1 :: (\Gamma) & \text{(by } \Lambda 2) \\
\Gamma \models \Lambda; \Theta_1, \Psi_a', \Psi_c', \Theta_2 :: (\Gamma; \Delta) & \text{(by } \Omega)
\end{array}$$

Well-formedness is maintained because $\Psi_c \notin \Theta$ and there is a corresponding $\text{unavail}(c_S)$ to the newly added Ψ_c'.

Other omitted cases follow a similar strategy as presented.

□

G Progress

Theorem 2. *If* $\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta)$ *then either:*

1. $\Lambda \to \Lambda'$ *for some* Λ' *or*

2. Λ *poised and one of:*

(a) $\Lambda; \Theta \rightarrow \Lambda'; \Theta'$ *for some* Λ' *and* Θ' *or*

(b) Θ *poised or*

(c) some $\Psi \in \Theta$ *is blocked*

Proof.

$$\Gamma \models \Lambda; \Theta :: (\Gamma; \Delta) \qquad \text{(by assumption)}$$
$$\Gamma \models \Lambda :: (\Gamma) \quad \Gamma \models \Theta :: (\Delta) \qquad \text{(by inversion on } \Omega)$$

for some Γ, Λ, Θ, and Δ.

We first show that either $\Lambda \rightarrow \Lambda'$ for some Λ' or that Λ is poised by induction on the derivation of $\Gamma \models \Lambda :: (\Gamma)$.

Case 1.

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Lambda 1$$

$(\cdot)$ is poised since there is no proc term.

Case 2.

$$\frac{\Gamma \models \Lambda_1 :: (\Gamma_1) \quad \Gamma \models \Lambda_2 :: (\Gamma_2)}{\Gamma \models \Lambda_1, \Lambda_2 :: (\Gamma_1, \Gamma_2)}\ \Lambda 2$$

Then either $\Lambda_1 \rightarrow \Lambda_1'$ or Λ_1 is poised by induction hypothesis, and similarly, either $\Lambda_2 \rightarrow \Lambda_2'$ or Λ_2 is poised by induction hypothesis. If both Λ_1 and Λ_2 are poised, then the concatenation Λ_1, Λ_2 is poised. Otherwise, we take the concatenation of the components that progresses. In particular, if $\Lambda_1 \rightarrow \Lambda_1'$ and Λ_2 is poised, $\Lambda_1, \Lambda_2 \rightarrow \Lambda_1', \Lambda_2$ (and similarly for the other two combinations).

Case 3.

$$\frac{\vdash (A_S', A_S, \top) \text{ dfsync} \quad \Gamma \vdash P :: (a_S{:}A_S')}{\Gamma \models \text{proc}(a_S, P) :: (a_S{:}A_S)}\ \Lambda 3$$

We proceed by case analysis on the syntactic form of P inferred from inversion on the appropriate typing rule on the derivation of $\Gamma \vdash P :: (a_S{:}A_S')$.

Subcase 1. $P = \text{fwd } a_S\ b_S$. This case requires a global substitution on the top level Λ. Since there is no ordering constraint on Λ, let $\Lambda = \text{proc}(a_S, \text{fwd } a_S\ b_S), \Lambda_1$ without loss of generality. Then by D-FWDSS,

$$\Lambda \rightarrow [a_S/b_S]\Lambda_1$$

Subcase 2. $P = x_S \leftarrow X_S \leftarrow \overline{b_S}; Q$, then by D-SPAWNSS,

$$\text{proc}(a_S, x_S \leftarrow X_S \leftarrow \overline{b_S}; Q) \rightarrow \text{proc}(a_S, [c_S/x_S]Q), \text{proc}(c_S, [c_S/x_S', \overline{b_S}/\overline{y_S'}]P) \quad (c \text{ fresh})$$

Subcase 3. $P = a_L \leftarrow \mathrm{acc}_S\ a_S; Q$, then $\mathrm{proc}(a_S, P)$ is poised by definition.
Case 4.

$$\frac{}{\Gamma \models \mathrm{unavail}(a_S) :: (a_S{:}\hat{A})}\ \Lambda 4$$

$\mathrm{unavail}(a_S)$ is poised since there is no proc term.

That concludes the first part of the proof. Now to show the second part, we will assume that Λ is poised and proceed by induction on the derivation of $\Gamma \models \Theta :: (\Delta)$ to show one of:

(a) $\Lambda; \Theta \to \Lambda'; \Theta'$ for some Λ' and Θ'

(b) Θ poised

(c) some $\Psi \in \Theta$ is strongly blocked

We will showcase the style of the proof along with the interesting cases.
Case 1.

$$\frac{}{\Gamma \models \cdot :: (\cdot)}\ \Theta 1$$

$(\cdot)$ is poised since there is no proc term.
Case 2.

$$\frac{b_S{:}\hat{B} \in \Gamma \quad b_S \leq A_L \quad \Gamma \models \Theta_1 :: (\Delta_1)}{\Gamma \models \mathrm{connect}(a_L, b_S), \Theta_1 :: (a : A_L, \Delta_1)}\ \Theta 2$$

By the induction hypothesis, Θ_1 either steps, is poised, or contains a Ψ that is strongly blocked.

If Θ_1 steps, then $\Lambda; \Theta_1 \to \Lambda'; \Theta_1'$ for some Λ' and Θ_1'. Then $\Lambda; \mathrm{connect}(a_L, b_S), \Theta_1 \to \Lambda'; \mathrm{connect}(a_L, b_S), \Theta_1'$.

If Θ_1 is poised, then $\mathrm{connect}(a_L, b_S), \Theta_1$ is poised because $\mathrm{connect}(-_L, -_S)$ is not a proc term.

Finally, if there is some $\Psi \in \Theta_1$ that is strongly blocked, then of course the same $\Psi \in (\mathrm{connect}(a_L, b_S), \Theta_1)$ is strongly blocked.
Case 3.

$$\frac{c_S{:}\hat{C} \in \Gamma \quad \vdash (C_L', C_L, \hat{C})\ \mathrm{dfsync} \quad \Gamma; \Delta_c \vdash P :: (c_L{:}C_L') \quad \Gamma \models \Theta_1 :: (\Delta_c, \Delta_1)}{\Gamma \models \mathrm{proc}(c_L, P), \Theta_1 :: (c : C_L, \Delta_1)}\ \Theta 3$$

By the induction hypothesis, Θ_1 either steps, is poised, or contains a Ψ that is strongly blocked. We first cover two of the cases:

If Θ_1 steps, then $\Lambda; \Theta_1 \to \Lambda'; \Theta_1'$ for some Λ' and Θ_1'. Then $\Lambda; \mathrm{proc}(c_L, P), \Theta_1 \to \Lambda'; \mathrm{proc}(c_L, P), \Theta_1'$.

If there is some $\Psi \in \Theta_1$ that is strongly blocked, then of course the same $\Psi \in (\text{proc}(c_L, P), \Theta_1)$ is strongly blocked.

For the final case, we will assume that Θ_1 is poised and proceed by case analysis on the derivation of $\Gamma; \Delta_c \vdash P :: (c_L{:}C'_L)$. Unlike in the first part, we make the step between identifying the appropriate typing rule and inferring the form of P explicit because some of the cases are more complicated. In the type judgement, we replace instantiated channel variables in the context such as x by actual channel names since they must already exist in the configuration.

Subcase 1. The form of P inferred from all linear right rules ($1R, \otimes R, \otimes R_S, \multimap R, \oplus R, \& R, \uparrow_L^L R, \downarrow_L^L R$) directly coincide with the definition of poised. For example, $1R$ implies that $P = \text{close } a_L$, which is poised, and so on. Since Θ_1 is poised, $\text{proc}(a_L, P), \Theta_1$ is poised.

Subcase 2.

$$\frac{\Gamma; \Delta'_c, b_L{:}B_L, y_L{:}A_L \vdash P :: (c_L{:}C'_L)}{\Gamma; \Delta'_c, b_L{:}A_L \otimes B_L \vdash y_L \leftarrow \text{recv } b_L; P :: (c_L{:}C'_L)} \otimes L$$

where $\Delta_c = \Delta'_c, b_L{:}A_L \otimes B_L$. Then $\Theta_1 = \Theta_2, \text{proc}(b_L, -), \Theta_3$ for some Θ_2 and Θ_3 (we know b_L is not provided by a connect term since connect terms offer channels of type $\uparrow_L^L D_L$). Since $\text{proc}(b_L, -)$ is poised and must offer a channel of type $A_L \otimes B_L$, it must be of form $\text{proc}(b_L, \text{send } b_L\ a_L; Q)$. Thus, by D-$\otimes$,

$$\Lambda; \text{proc}(c_L, y_L \leftarrow \text{recv } b_L; P), \Theta_2, \text{proc}(b_L, \text{send } b_L\ a_L; Q), \Theta_3 \rightarrow \Lambda; \text{proc}(c_L, [a_L/y_L]P), \Theta_2, \text{proc}(b_L, Q), \Theta_3$$

All the remaining linear left rules except $\uparrow_L^L L$ and $\uparrow_L^L R$ ($1L, \multimap L, \multimap L_S, \oplus L, \& L$) follow a similar pattern.

Subcase 3.

$$\frac{\hat{A} \leq \uparrow_L^S A_L \quad \Gamma, a_S{:}\hat{A}; \Delta, x_L{:}A_L \vdash P :: (c_L{:}C'_L)}{\Gamma, a_S{:}\hat{A}; \Delta_c \vdash x_L \leftarrow \text{acq}_S\ a_S; P :: (c_L{:}C'_L)} \uparrow_L^S L$$

Since Λ is poised, either $\Lambda = \text{unavail}(a_S), \Lambda_1$ or $\Lambda = \text{proc}(a_S, x_L \leftarrow \text{acc}_S\ a_S; Q), \Lambda_1$ for some Λ_1. In the first case, $\text{proc}(c_L, a_L \leftarrow \text{acq}_S\ a_S; P)$ is strongly blocked, so we are done. In the second case, by D-$\uparrow_L^S$, we have

$$\begin{aligned} & \text{proc}(a_S, x_L \leftarrow \text{acc}_S\ a_S; Q), \Lambda_1; \text{proc}(c_L, a_L \leftarrow \text{acq}_S\ a_S; P), \Theta_1 \\ \rightarrow\ & \text{unavail}(a_S), \Lambda_1; \text{proc}(c_L, [a_L/x_L]P), \text{proc}(a_L, [a_L/x_L]Q), \Theta_1 \end{aligned}$$

Subcase 4.

$$\frac{\Gamma, x_S{:}A_S; \Delta'_c \vdash P :: (c_L{:}C'_L)}{\Gamma; \Delta'_c, a_L{:}\downarrow_L^S A_S \vdash x_S \leftarrow \text{rel}_S\ a_S; P :: (c_L{:}C'_L)} \downarrow_L^S L$$

where $\Delta_c = \Delta'_c, a_L{:}\downarrow_L^S A_S$. Then $\Theta_1 = \Theta_2, \text{proc}(a_L, -), \Theta_3$ for some Θ_2 and Θ_3. Since there is a $\text{proc}(a_L, -)$ in the linear configuration, by well-formedness condition, there must be a corresponding

$\text{unavail}(a_S) \in \Lambda$, so $\Lambda = \text{unavail}(a_S), \Lambda_1$. Furthermore, since Θ_1 is poised, the proc term must be of form $\text{proc}(a_L, x_S \leftarrow \text{det}_S\ a_S; Q)$. By D-$\downarrow_L^S$, we have

$$\begin{array}{rl} & \text{unavail}(a_S), \Lambda_1; \text{proc}(c_L, x_S \leftarrow \text{rel}_S\ a_S; P), \Theta_2, \text{proc}(a_L, x_S \leftarrow \text{det}_S\ a_S; Q), \Theta_3 \\ \to & \text{proc}(a_S, [a_S/x_S]Q), \Lambda_1; \text{proc}(c_L, [a_L/x_L]P), \Theta_2, \Theta_3 \end{array}$$

Subcase 5.

$$\frac{\Gamma; \Delta_c', x_L{:}A_L \vdash P :: (c_L{:}C_L')}{\Gamma; \Delta_c', a_L{:}\uparrow_L^L A_L \vdash x_L \leftarrow \text{acq}_L\ a_L; P :: (c_L{:}C_L')} \uparrow_L^L L$$

where $\Delta_c = \Delta_c', a_L{:}\uparrow_L^L A_L$. Then $\Theta_1 = \Theta_2, \Psi_a, \Theta_3$ where Ψ_a is either of form $\text{connect}(a_L, b_S)$ for some b_S or $\text{proc}(a_L, -)$. In the latter case, we appeal to the term being poised and the proof proceeds like the other left rules. In the former case, there must be a term in Λ that provides b_S. Since Λ is poised, either $\Lambda = \text{unavail}(b_S), \Lambda_1$ or $\Lambda = \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q), \Lambda_1$. In the former case, we can conclude that $\text{proc}(c_L, -)$ is strongly blocked, so we are done. In the latter case, by D-$\uparrow_L^S 2$, we have

$$\begin{array}{rl} & \text{proc}(b_S, x_L \leftarrow \text{acc}_S\ b_S; Q), \Lambda_1; \text{proc}(c_L, x_L \leftarrow \text{acq}_L\ a_L; P), \Theta_2, \text{connect}(a_L, b_S), \Theta_3 \\ \to & \text{unavail}(b_S), \Lambda_1; \text{proc}(c_L, [b_L/x_L]P), \text{proc}(b_L, [b_L/x_L]Q), \Theta_2, \Theta_3 \end{array}$$

Subcase 6.

$$\frac{\Gamma; \Delta_c', x_L{:}A_L \vdash P :: (c_L{:}C_L')}{\Gamma; \Delta_c', a_L{:}\downarrow_L^L A_L \vdash x_L \leftarrow \text{rel}_L\ a_L; P :: (c_L{:}C_L')} \downarrow_L^L L$$

where $\Delta_c = \Delta_c', a_L{:}\downarrow_L^L A_L$. Then $\Theta_1 = \Theta_2, \text{proc}(a_L, -), \Theta_3$. Since Θ_1 is poised, there are two possible forms of $\text{proc}(a_L, -)$. If we have $\text{proc}(a_L, x_L \leftarrow \text{det}_L\ a_L; Q)$, then we appeal to the term being poised like the other left rules. If we instead have $\text{proc}(a_L, x_S \leftarrow \text{det}_S\ a_S; Q)$, then we first identify that $\Lambda = \text{unavail}(a_S), \Lambda_1$ for some Λ_1 by the well-formedness condition. By D-$\downarrow_L^S 2$, we have

$$\begin{array}{rl} & \text{unavail}(a_S), \Lambda_1; \text{proc}(c_L, x_L \leftarrow \text{rel}_L\ a_L; P), \Theta_2, \text{proc}(a_L, x_S \leftarrow \text{det}_S\ a_S; Q), \Theta_3 \\ \to & \text{proc}(a_S, [a_S/x_S]Q), \Lambda_1; \text{proc}(c_L, [b_L/x_L]P), \text{connect}(b_L, a_S), \Theta_2, \Theta_3 \quad (b\ \ \text{fresh}) \end{array}$$

□