A Domain Semantics for Higher-Order Recursive Processes

RYAN KAVANAGH, Carnegie Mellon University, United States of America

The polarized SILL programming language [Pfenning and Griffith 2015; Toninho et al. 2013] uniformly integrates functional programming and session-based concurrency. It supports recursion, asynchronous and synchronous communication, and higher-order programs that communicate processes. We build on work of Atkey [2017] to give the first denotational semantics for a language with these features. Session types are interpreted as domains of bidirectional communications. We use polarity to naturally relate these to pairs of domains of unidirectional communications. Processes denote continuous functions between domains of unidirectional communication. Semantic composition of processes is given by a novel fixed-point operator that validates identities similar to dagger operations [Bloom and Ésik 1996]. We show that our semantics validates many expected program equivalences.

Additional Key Words and Phrases: session types, denotational semantics, recursion, higher-order processes

1 INTRODUCTION

The proofs-as-programs correspondence between linear logic and the session-typed π-calculus is the foundation of many programming languages for message-passing concurrency. This logical underpinning has given rise to an arsenal of techniques for reasoning about processes [Pérez et al. 2012, 2014; Toninho 2015]. Denotational semantics [Atkey 2017] and game semantics [Castellan and Yoshida 2019] have further enriched our understanding. However, giving a denotational semantics to higher-order session-typed languages with recursion has remained a difficult problem.

To illustrate the difficulty, consider a process F that flips bits in a bit stream. A bit stream is an infinite sequence of labels `0` or `1` sent over a channel. This protocol is specified by a recursive session type "`bits`".[1]

```
b : bits |- F :: f : bits
f <- F <- b = case b { 0 => f.1; f <- F <- b
                     | 1 => f.0; f <- F <- b }
```

This process cases on the label received over b and sends the complementary label over f. Subsequent communication on f is provided by the tail call to F using b as input. Chaining two copies of F together should have the same effect as simply forwarding b over f. In other words, we expect the following two processes to be equivalent:

```
b : bits |- t <- F <- b; f <- F <- t :: f : bits
b : bits |- f <- b :: f : bits
```

What do we mean by equivalence? We expect processes to be equivalent whenever we cannot observe any meaningful differences. In the case of processes, termination is not meaningful. Indeed, the forwarding process globally identifies the two channels and immediately terminates under typical operational semantics, while the bit flipping processes never terminate. In contrast, a process's interaction with its environment *is* meaningful: what messages does the process send in response to messages it receives? Operational semantics for session-typed processes generally satisfy a confluence property stating that their input-output behaviour is deterministic. These facts suggest that F denotes a function $[\![F]\!]$ from bit streams on b to bit streams on f.

The processes-as-functions interpretation immediately raises many questions. How should we interpret the type `bits`? The process providing the bit stream on b could get stuck and only send a

[1] Explicitly, `bits` $= \rho\beta.\, \oplus\{\mathtt{0} : \beta, \mathtt{1} : \beta\}$, where $\rho\alpha.A$ forms recursive types and $\oplus\{l : A_l\}_{l \in L}$ forms internal choice types.

Author's address: Ryan Kavanagh, Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213-3891, United States of America, rkavanagh@cs.cmu.edu.

finite prefix of this bit stream. This suggests the interpretation of `bits` should also contain finite prefixes. How should $[\![F]\!]$ handle these finite prefixes? Computationally, $[\![F]\!]$ should be monotone: a longer input prefix should result in a longer output prefix. It should also be continuous: $[\![F]\!]$'s behaviour on entire streams should be consistent with its behaviour on their finite prefixes. The ordering on `bits` is clear, but how should we define the ordering for other types?

The processes-as-functions interpretation is further complicated by the fact that communication on channels is bidirectional. This implies that each channel can serve as both an "input" and an "output" channel. If a process P provides a channel c of type A, then the session type A specifies a communication protocol on the channel c. How can we decompose the interpretation $[\![A]\!]$ of A into the "inputs" and "outputs" for P? Is there a natural relation between $[\![A]\!]$ and this decomposition?

Finally, what does it mean to compose processes? If processes denote functions, then function composition is insufficient to capture bidirectional communication. Instead, we need a composition where the "output" of each process on the common channel feeds into the "input" of the other. How do we define this circular composition and what are its properties? Communication can be synchronous or asynchronous. How do we reflect this distinction with functions?

Our thesis is that a domain-theoretic semantics elucidates the structure of higher-order session-typed languages with recursion. Domains are well-understood mathematical structures and we use their rich theory to study these languages. A denotational semantics induces a notion of semantic equivalence. It is automatically compositional because denotational semantics are defined by recursion on the structure of programs. Using this semantic equivalence, we discover that some equivalences hold only when we remove recursion from our language.

We give a domain-theoretic semantics for a polarized extension of the language SILL [Pfenning and Griffith 2015; Toninho et al. 2013]. It integrates functional and message-passing concurrent programming using a contextual monad. Recursive processes are implemented using the functional layer and process communication is asynchronous. Synchronous communication is encoded using explicit shifts of polarity. Our semantics provides three main contributions:

(1) **An order-theoretic analysis of polarized session types.** We give a high-level overview of our language and its semantics in section 2. We show that the decomposition of bidirectional communication into "input" and "output" is intrinsically linked to polarity. We show that this decomposition is given by a natural embedding-projection pair.
(2) **A denotational semantics for our language.** In section 3, we give the details of our semantics. We interpret types as domains, and processes and terms as continuous functions. We explain how the types' decomposition into their input and output aspects makes operational intuitions about clients and providers concrete. We interpret definitional equality for types as natural isomorphisms. We show that these interpretations validate expected equivalences.
(3) **A recursive notion of function composition.** In section 4, we develop the theory of our process composition operator. We show that it validates identities similar to those for the dagger operation [Bloom and Ésik 1996; Ésik 2009] of fixed-point theory. We give identities useful for computing semantic equivalences.

We illustrate our semantics by studying processes operating on bit streams in section 5.

2 OVERVIEW

The polarized SILL programming language cohesively integrates functional computation and message-passing concurrent computation. Its functional layer is the simply-typed λ-calculus with a fixed-point operator and a call-by-value evaluation semantics. Encapsulated open processes are a base type. A hypothetical judgment $\Psi \Vdash M : \tau$ means the functional term M has functional type τ under the structural context Ψ of functional variables $x_i : \tau_i$.

The semantics of the functional layer is standard [Crole 1993; Gunter 1992; Reynolds 2009; Tennent 1995]. We interpret functional types τ as a Scott domain (explicitly, an ω-algebraic bounded-complete dcpo) $[\![\tau]\!]$. A structural context Ψ is interpreted as the Ψ-indexed product $[\![\Psi]\!] = \prod_{x \in \Psi} [\![\tau]\!]$. A well-typed functional term $\Psi \Vdash M : \tau$ then denotes a continuous function

$$[\![\Psi \Vdash M : \tau]\!] : [\![\Psi]\!] \to [\![\tau]\!]$$

in the category ω-**aBC** of Scott domains and continuous functions.

The process layer arises from a proofs-as-programs correspondence between intuitionistic linear logic and the session-typed π-calculus [Caires and Pfenning 2010]. A session type A describes a protocol for communicating over a channel. A process P provides a service A on a channel c while using zero or more services A_i on channels c_i. The used services form a linear context $\Delta = c_1 : A_1, \ldots, c_n : A_n$. The process P can also use values from the functional layer. These are abstracted by a structural context Ψ of functional variables. These data are captured by the hypothetical judgment $\Psi \,;\, \Delta \vdash P :: c : A$.

We cannot interpret processes $\Psi \,;\, \Delta \vdash P :: c : A$ in the same manner as functional terms, that is, as functions $[\![\Psi]\!] \times [\![\Delta]\!] \to [\![A]\!]$. This is because of the fundamental difference between *variables* and *channel names*. A variable $x : \tau$ in the context Ψ stands for a value of type τ. The channels $\Delta, c : A$ do not stand for a values, but bidirectional communications obeying the channel's type. To capture the input-output behaviour of P, we view each bidirectional channel as a pair of unidirectional channels: one for input and one for output. We then interpret P as a continuous function

$$[\![\Psi \,;\, \Delta \vdash P :: c : A]\!] : [\![\Psi]\!] \to [\mathsf{inputOn}(\Delta, c : A) \to \mathsf{outputOn}(\Delta, c : A)],$$

where we use the notation $[\cdot \to \cdot]$ for function spaces.

To make this precise, we use the relationship between the polarity of a session type and the direction of communication along a channel of that type [Pfenning and Griffith 2015]. Session types can be partitioned as *positive* or *negative*. When looking at a judgment $\Psi \,;\, \Delta \vdash P :: c : A$, we can imagine "positive information" as flowing left-to-right and "negative information" as flowing right-to-left. Indeed, when the provided service A is positive, communication on c is sent by P; when A is negative, it is received by P. Symmetrically, when a used service A_i is positive, communication on c_i is received by P; when A_i is negative, it is sent by P. As P executes, the type of a channel $b : B$ in $\Delta, c : A$ changes, sometimes becoming a positive subphrase of B, sometimes a negative subphrase B. It is this evolution of a channel's type that causes bidirectionality.

By looking at a session type through negative- and positive-coloured glasses, we see its positive and negative aspects. Given a channel $b : B$, the positive aspect of B prescribes the left-to-right communication along b, while the negative aspect of B prescribes the right-to-left communication. This leads to three interpretations, each a Scott domain. The canonical interpretation $[\![B]\!]$ captures bidirectional communications of type B, while the negative and positive aspects $[\![B]\!]^-$ and $[\![B]\!]^+$ contain the "negative" and "positive" portions of the bidirectional communication. Proposition 2.1 shows that these three interpretations are intimately and naturally related. We also show in section 3 that there are strong computational motivations for these polarized aspects.

By so leveraging polarity, we interpret a process P as a continuous function

$$[\![\Psi \,;\, \Delta \vdash P :: c : A]\!] : [\![\Psi]\!] \to [[\![\Delta]\!]^+ \times [\![c : A]\!]^- \to [\![\Delta]\!]^- \times [\![c : A]\!]^+]$$

in ω-**aBC**, where $[\![b_1 : B_1, \ldots, b_n : B_n]\!]^p$ denotes the indexed product $\left(b_1^p : [\![B_1]\!]^p\right) \times \cdots \times \left(b_n^p : [\![B_n]\!]^p\right)$ for $p \in \{-, +\}$. This interpretation satisfies the desiderata given in the introduction: continuity ensures that "better input" results in "better output". Operational semantics for session-typed languages typically satisfy a confluence property. Functionality builds-in this determinism for free.

Process composition $b \leftarrow P;\ Q$ arises from the proofs-as-processes interpretation of the (Cut) rule in intuitionistic linear logic:

$$\frac{\Psi\ ;\ \Delta_1 \vdash P :: b : B \quad \Psi\ ;\ b : B, \Delta_2 \vdash Q :: c : C}{\Psi\ ;\ \Delta_1, \Delta_2 \vdash b \leftarrow P;\ Q :: c : C}\ (\textsc{Cut})$$

Operationally, the process $\Psi\ ;\ \Delta_1, \Delta_2 \vdash b \leftarrow P;\ Q :: c : C$ spawns two processes $\Psi\ ;\ \Delta_1 \vdash P :: b : B$ and $\Psi\ ;\ b : B, \Delta_2 \vdash Q :: c : C$ communicating along the shared channel b of type B. Our interpretation of processes is recursively defined on the typing derivation. This means that for each $u \in [\![\Psi]\!]$ we must define a continuous function

$$[\![\Psi\ ;\ \Delta_1, \Delta_2 \vdash b \leftarrow P;\ Q :: c : C]\!]u : [\![\Delta_1, \Delta_2]\!]^+ \times [\![c : C]\!]^- \to [\![\Delta_1, \Delta_2]\!]^- \times [\![c : C]\!]^+$$

given continuous functions

$$[\![\Psi\ ;\ \Delta_1 \vdash P :: b : B]\!]u : [\![\Delta_1]\!]^+ \times [\![b : B]\!]^- \to [\![\Delta_1]\!]^- \times [\![b : B]\!]^+,$$
$$[\![\Psi\ ;\ b : B, \Delta_2 \vdash Q :: c : C]\!]u : [\![b : B]\!]^+ \times [\![\Delta_2]\!]^+ \times [\![c : C]\!]^- \to [\![b : B]\!]^- \times [\![\Delta_2]\!]^- \times [\![c : C]\!]^+.$$

Consider input $(\delta_1^+, \delta_2^+, c^-) \in [\![\Delta_1, \Delta_2]\!]^+ \times [\![c : C]\!]^-$ to their composition $b \leftarrow P;\ Q$. It represents the totality of a communication of those polarities on those channels. As the processes P and Q process this input, they produce output on b^+ and b^- that must be fed back into Q and P. To capture this, we consider the sequence f_n of functions inductively defined as

$$f_0(\delta_1^+, \delta_2^+, c^-) = \left([\![P]\!]u(\delta_1^+, \bot), [\![Q]\!]u(\bot, \delta_2^+, c^-)\right),$$
$$f_{n+1}(\delta_1^+, \delta_2^+, c^-) = \left([\![P]\!]u(\delta_1^+, b_n^-), [\![Q]\!]u(b_n^+, \delta_2^+, c^-)\right),$$

where

$$\left((\delta_1^-)_n, b_n^+\right), \left(b_n^-, (\delta_2^-)_n, c_n^+\right)\big) = f_n(\delta_1^+, \delta_2^+, c^-).$$

This sequence converges to a function f. Let $\left(\left(\delta_1^-, b^+\right), \left(b^-, \delta_2^-, c^+\right)\right) = f(\delta_1^+, \delta_2^+, c^-)$. The pair (b^+, b^-) captures the entire bidirectional communication on b between P and Q, and $(\delta_1^-, \delta_2^-, c^+)$ is their output on $\Delta_1, \Delta_2, c : C$. We let $[\![\Psi\ ;\ \Delta_1, \Delta_2 \vdash b \leftarrow P;\ Q :: c : C]\!]u(\delta_1^+, \delta_2^+, c^-) = (\delta_1^-, \delta_2^-, c^+)$.

This fixed point is an instance of our novel "double-dagger" fixed-point construction that takes a continuous function $f : A \times X \to B \times X$ to a continuous function $f_X^\ddagger : A \to B$. Fixed-point constructions can be difficult to reason about. Fortunately, our construction satisfies many identities similar to those satisfied by dagger operations [Bloom and Ésik 1996; Ésik 2009] in fixed-point theory. We repeatedly use these identities below to simplify reasoning about process composition.

We build on standard domain-theoretic techniques to interpret session types and functional types. Recursive types are interpreted as solutions to domain equations. To ensure these solutions exist, we must be careful when defining and interpreting our types. Our closed session types are all contractive [Pierce 2002, p. 300], that is, for any subphrase of the form $\rho\alpha.\rho\alpha_1.\cdots\rho\alpha_n.A$, A is not α. The judgments $A\ \mathrm{type}_{\mathrm{s}}^p$ and $A\ \mathrm{ctype}_{\mathrm{s}}^p$ mean A is a session type or a contractive session type with polarity p. Open types are captured using hypothetical judgments $\Xi \vdash A\ \mathrm{type}_{\mathrm{s}}^p$ and $\Xi \vdash A\ \mathrm{ctype}_{\mathrm{s}}^p$, where $\Xi = \alpha_1\ \mathrm{type}_{\mathrm{s}}^{p_1}, \ldots, \alpha_n\ \mathrm{type}_{\mathrm{s}}^{p_n}$ is a structural context of polarized type variables. We abbreviate these judgments as $\Xi \vdash A$ where no ambiguity arises.

We interpret hypothetical judgments $\Xi \vdash A$ as locally continuous functors on a category of domains. Let ω-$\mathbf{aBC}_{\bot!}$ be the subcategory of ω-$\mathbf{aBC}$ whose morphisms are strict functions. Let $\mathbf{M}$ be the subcategory of ω-$\mathbf{aBC}_{\bot!}$ whose morphisms are also meet-semilattice homomorphisms (meet-homomorphisms). Explicitly, the morphisms of $\mathbf{M}$ are continuous functions $f : A \to B$ such that $f(\bot_A) = \bot_B$ and $f(x \sqcap y) = f(x) \sqcap f(y)$ for all $x, y \in A$. The restriction to meet-homomorphisms is needed for order-theoretic reasons described in section 3.8. The category $\mathbf{M}$ inherits from ω-$\mathbf{aBC}_{\bot!}$ the structure required to solve domain equations. This fact follows from techniques by Smyth and

Plotkin [1982] and the fact that the meet operator $\sqcap : D \times D \to D$ is continuous for bounded-complete domains D. The category $\mathbf{M}$ has products but it does not have coproducts. Let $\mathbf{M}^\Xi$ be the Ξ-indexed product $\prod_{\alpha \in \Xi} \mathbf{M}$. The canonical interpretation $[\![\Xi \vdash A]\!]$ and polarized aspects $[\![\Xi \vdash A]\!]^-$ and $[\![\Xi \vdash A]\!]^+$ are then locally continuous functors from $\mathbf{M}^\Xi$ to $\mathbf{M}$.

We relate these three type interpretations using embedding-projection-pairs. Given two morphisms $f, g : A \to B$ in ω-**aBC**, we say $f \sqsubseteq g$ if for all $a \in A$, $f(a) \sqsubseteq g(a)$ in B. A pair of morphisms $e : A \to B$ and $p : B \to A$ forms an embedding-projection pair or **e-p-pair** (e, p) if $p \circ e = \mathrm{id}_A$ and $e \circ p \sqsubseteq \mathrm{id}_B$. In this case, we say A is a **subdomain** of B: (e, p) specifies a copy of A at the bottom of B. We say e is an **embedding** and p is a **projection**. The morphisms e and p uniquely determine each other and p is always a meet-homomorphism [Abramsky and Jung 1995, Propositions 3.1.10 and 3.1.13]. Given an e-p-pair (f, g), we may write f^p for g and g^e for f. When A is a closed session type, the canonical interpretation $[\![A]\!]$ is a subdomain of $[\![A]\!]^- \times [\![A]\!]^+$.

To relate the interpretations of open types, we generalize from single morphisms to natural transformations. Given functors F and G into ω-**aBC**, we say a natural transformation $\epsilon : F \to G$ is a **natural pointwise-embedding** if each component $\epsilon_C : FC \to GC$ is an embedding; when each component is a projection, ϵ is a **natural pointwise-projection**. A natural pointwise-embedding ϵ is a **natural embedding** when the family of projections $\{\epsilon_C^p : GC \to FC\}$ is a natural transformation $G \to F$. The definition of **natural projection** is symmetric. When ϵ is a natural embedding with associated projection π, we say (ϵ, π) is a **natural e-p-pair**. The general relationship between interpretations of open types is given by proposition 2.1.

Proposition 2.1. *If J is a derivable hypothetical judgment of the form $\Xi \vdash A\ \mathsf{type_s}$ or $\Xi \vdash A\ \mathsf{ctype_s}$, then there exist strict continuous natural transformations*

$$\begin{aligned} \mathsf{merge}_J &: [\![J]\!]^- \times [\![J]\!]^+ \to [\![J]\!], \\ \pi_J^+ &: [\![J]\!] \to [\![J]\!]^+, \text{ and} \\ \pi_J^- &: [\![J]\!] \to [\![J]\!]^- \end{aligned}$$

that form a natural e-p-pair $\left(\langle \pi_J^-, \pi_J^+ \rangle, \mathsf{merge}_J\right) : [\![J]\!] \xrightarrow{\langle \pi_J^-, \pi_J^+ \rangle} [\![J]\!]^- \times [\![J]\!]^+ \xrightarrow{\mathsf{merge}_J} [\![J]\!]$.

Recursive types in our language are equirecursive instead of isorecursive. This means that instead of using explicit fold and unfold constructions within the language, we deem a recursive type to be definitionally equal to its unfolded form. The judgment $\Xi \vdash A \equiv A'$ means that the types $\Xi \vdash A$ and $\Xi \vdash A'$ are definitionally equal. We interpret this judgment as a natural isomorphism $[\![\Xi \vdash A \equiv A']\!] : [\![\Xi \vdash A]\!] \cong [\![\Xi \vdash A']\!]$. Proposition 2.2 shows that we can transfer these natural isomorphisms from the canonical interpretations to the polarized aspects.

Proposition 2.2. *If $\psi = [\![\Xi \vdash A \equiv B]\!] : [\![\Xi \vdash A\ \mathsf{type_s}]\!] \cong [\![\Xi \vdash B\ \mathsf{type_s}]\!]$, then there exist natural isomorphisms $\psi^+ : [\![\Xi \vdash A\ \mathsf{type_s}]\!]^+ \cong [\![\Xi \vdash B\ \mathsf{type_s}]\!]^+$ and $\psi^- : [\![\Xi \vdash A\ \mathsf{type_s}]\!]^- \cong [\![\Xi \vdash B\ \mathsf{type_s}]\!]^-$ that commute with the continuous e-p-pair of proposition 2.1, i.e., the following diagram commutes:*

$$\begin{array}{ccccc} [\![\Xi \vdash A\ \mathsf{type_s}]\!] & \xrightarrow{\langle \pi_A^-, \pi_A^+ \rangle} & [\![\Xi \vdash A\ \mathsf{type_s}]\!]^- \times [\![\Xi \vdash A\ \mathsf{type_s}]\!]^+ & \xrightarrow{\mathsf{merge}_A} & [\![\Xi \vdash A\ \mathsf{type_s}]\!] \\ \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi^- \times \psi^+} & & \downarrow{\scriptstyle \psi} \\ [\![\Xi \vdash B\ \mathsf{type_s}]\!] & \xrightarrow[\langle \pi_B^-, \pi_B^+ \rangle]{} & [\![\Xi \vdash B\ \mathsf{type_s}]\!]^- \times [\![\Xi \vdash B\ \mathsf{type_s}]\!]^+ & \xrightarrow[\mathsf{merge}_B]{} & [\![\Xi \vdash B\ \mathsf{type_s}]\!]. \end{array} \tag{1}$$

3 SEMANTIC CLAUSES

We define the denotations of judgments by recursion on their derivation. A denotational semantics induces a notion of semantic equivalence. We illustrate our semantics by proving equivalences.

Definition 3.1 (Semantic equivalence). Processes $\Psi \mathrel{;} \Delta \vdash P :: c : A$ and $\Psi \mathrel{;} \Delta \vdash Q :: c : A$ are equivalent, $P \equiv Q$, if $[\![\Psi \mathrel{;} \Delta \vdash P :: c : A]\!] = [\![\Psi \mathrel{;} \Delta \vdash Q :: c : A]\!]$. Functional terms $\Psi \Vdash M : \tau$ and $\Psi \Vdash N : \tau$ are equivalent, $M \equiv N$, if $[\![\Psi \Vdash M : \tau]\!] = [\![\Psi \Vdash N : \tau]\!]$.

When defining the semantic clause for a process $\Psi \mathrel{;} \Delta \vdash P :: c : A$, we let the metavariable $u \in [\![\Psi]\!]$ range over functional environments. We make frequent use of indexed products and we may make their indices explicit to reduce ambiguity. We use the notation $(d_1 : D_1) \times \cdots \times (d_n : D_n)$ to denote the product of the D_i indexed by the d_i. Given $\delta_i \in D_i$ for $1 \leq i \leq n$, we write $(d_1 : \delta_1, \ldots, d_n : \delta_n)$ for an element of this product. Given an indexed product $\prod_{i \in I} D_i$ and a subset $J \subseteq I$, we write π^I_J or π_J for the projection $\prod_{i \in I} D_i \to \prod_{j \in J} D_j$.

Let $F, H : \mathbf{D} \to \mathbf{E}$ and $G, I : \mathbf{C} \to \mathbf{D}$ be functors, and let $\eta : F \to H$ and $\rho : G \to I$ be natural transformations. They form natural transformations $\eta G : FG \to HG$ and $F\rho : FG \to FI$ whose components at $C \in \mathbf{C}$ are $(\eta G)_C = \eta_{GC}$ and $(F\rho)_C = F(\rho_C)$. The **horizontal composition** $\eta * \rho : FG \to HI$ is given by the equal natural transformations $\eta I \circ F\rho$ and $H\rho \circ \eta G$.

3.1 Structural rules

The identity rule (Fwd) of linear logic corresponds to forwarding.

$$\frac{}{\Psi \mathrel{;} a : A \vdash b \leftarrow a :: b : A} \text{ (Fwd)}$$

It denotes the function that forwards the positive data on y^+ to x^+ and the negative data on x^- to y^-:

$$\begin{aligned} &[\![\Psi \mathrel{;} y : A \vdash x \leftarrow y :: x : A]\!] u : [\![y : A]\!]^+ \times [\![x : A]\!]^- \to [\![y : A]\!]^- \times [\![x : A]\!]^+ \\ &[\![\Psi \mathrel{;} y : A \vdash x \leftarrow y :: x : A]\!] u (y^+, x^-) = (x^-, y^+). \end{aligned} \tag{2}$$

The (Cut) rule corresponds to process composition.

$$\frac{\Psi \mathrel{;} \Delta_1 \vdash P :: b : B \quad \Psi \mathrel{;} b : B, \Delta_2 \vdash Q :: c : C}{\Psi \mathrel{;} \Delta_1 \Delta_2 \vdash b \leftarrow P \mathrel{;}\ Q :: c : C} \text{ (Cut)}$$

As motivated above, we use the double-dagger operation from section 4 to fix the component $b^- \times b^+$:

$$\begin{aligned} &[\![\Psi \mathrel{;} \Delta_1, \Delta_2 \vdash b \leftarrow P \mathrel{;}\ Q :: c : C]\!] u : [\![\Delta_1, \Delta_2]\!]^+ \times [\![c : C]\!]^- \to [\![\Delta_1, \Delta_2]\!]^- \times [\![c : C]\!]^+ \\ &[\![\Psi \mathrel{;} \Delta_1, \Delta_2 \vdash b \leftarrow P \mathrel{;}\ Q :: c : C]\!] u \\ &= \left([\![\Psi \mathrel{;} \Delta_1 \vdash P :: b : B]\!] u \times [\![\Psi \mathrel{;} b : B, \Delta_2 \vdash Q :: c : C]\!] u\right)^{\ddagger}_{b^- \times b^+}. \end{aligned} \tag{3}$$

The unidirectional communications $(b^-, b^+) \in [\![B]\!]^- \times [\![B]\!]^+$ can be recovered using proposition 4.1. We can reassemble (b^-, b^+) into a bidirectional communication in $[\![B]\!]$ using merge_B.

Semantically, we expect process composition to be associative. Proposition 3.2 shows that this is the case.

Proposition 3.2. *For all processes* $\Psi \mathrel{;} \Delta_1 \vdash P_1 :: c_1 : C_1$, $\Psi \mathrel{;} c_1 : C_1, \Delta_2 \vdash P_2 :: c_2 : C_2$, *and* $\Psi \mathrel{;} c_2 : C_2, \Delta_3 \vdash P_3 :: c_3 : C_3$, *we have* $c_1 \leftarrow P_1 \mathrel{;}\ (c_2 \leftarrow P_2 \mathrel{;}\ P_3) \equiv c_2 \leftarrow (c_1 \leftarrow P_1 \mathrel{;}\ P_2) \mathrel{;}\ P_3$.

Proof. We abbreviate the typing judgments by their associated process. By propositions 4.2 and 4.4 we have for all $u \in [\![\Psi]\!]$:

$$\begin{aligned}
&[\![c_1 \leftarrow P_1;\ (c_2 \leftarrow P_2;\ P_3)]\!]u \\
&= \left([\![P_1]\!]u \times \left([\![P_2]\!]u \times [\![P_3]\!]u\right)^{\ddagger}_{c_2^+ \times c_2^-}\right)^{\ddagger}_{c_1^- \times c_1^+} \\
&= \left([\![P_1]\!]u \times [\![P_2]\!]u \times [\![P_3]\!]u\right)^{\ddagger}_{(c_2^- \times c_2^+) \times (c_1^- \times c_1^+)} \\
&= \left(\left([\![P_1]\!]u \times [\![P_2]\!]u\right)^{\ddagger}_{c_1^- \times c_1^+} \times [\![P_3]\!]u\right)^{\ddagger}_{c_2^- \times c_2^+} \\
&= [\![c_2 \leftarrow (c_1 \leftarrow P_1;\ P_2);\ P_3]\!]u.
\end{aligned}$$

□

In the proof of proposition 3.2, we used propositions 4.2 and 4.4 to collapse a nested fixed point to a single fixed point. In general, they can reduce the number of fixed points we must compute.

3.2 Closing and waiting on channels

A provider of type $\mathbf{1}$ terminates by sending the close message. A client wait $c; P$ waits on the channel c of type $\mathbf{1}$ until it receives the close message, and then it continues as P. The typing rules are:

$$\frac{}{\Xi \vdash \mathbf{1}\ \mathsf{ctype}_\mathsf{s}^+}\ (\mathrm{C}\mathbf{1}) \qquad \frac{}{\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}}\ (\mathbf{1}R) \qquad \frac{\Psi\ ;\ \Delta \vdash P :: a : A}{\Psi\ ;\ \Delta, c : \mathbf{1} \vdash \mathsf{wait}\ c; P :: a : A}\ (\mathbf{1}L)$$

The close message is the only message that can be sent on a channel of type $\mathbf{1}$. We interpret (C$\mathbf{1}$) as the constant functor onto the two-element pointed domain $\{*\}_\perp = \{\perp \sqsubseteq *\}$. The top element $*$ corresponds to the close message, while the bottom element represents the absence of communication. All communication on a channel of type $\mathbf{1}$ is positive. As a result, its positive aspect is its canonical interpretation. Its negative aspect is the constant functor onto the terminal object $\top_\mathbf{M} = \{\perp\}$:

$$[\![\Xi \vdash \mathbf{1}\ \mathsf{ctype}_\mathsf{s}]\!] = \lambda\xi.\{*\}_\perp, \tag{4}$$

$$[\![\Xi \vdash \mathbf{1}\ \mathsf{ctype}_\mathsf{s}]\!]^+ = [\![\Xi \vdash \mathbf{1}\ \mathsf{ctype}_\mathsf{s}]\!], \tag{5}$$

$$[\![\Xi \vdash \mathbf{1}\ \mathsf{ctype}_\mathsf{s}]\!]^- = \lambda\xi.\top_\mathbf{M}. \tag{6}$$

The relating natural transformations for proposition 2.1 are determined by $\pi_\mathbf{1}^+ = \mathrm{id}$ and $\pi_\mathbf{1}^- = \top_{[\![\mathbf{1}]\!]}$, the unique morphism given by terminality.

In our asyncronous setting, the process close c does not need to wait for a client before sending the close message. We interpret ($\mathbf{1}R$) as the constant function that sends the close message $*$ on c^+:

$$\begin{aligned}
&[\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u : [\![c : \mathbf{1}]\!]^- \to [\![c : \mathbf{1}]\!]^+ \\
&[\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u(c^- : \perp) = (c^+ : *).
\end{aligned} \tag{7}$$

The process wait $c; P$ blocks until it receives the close message. If it receives no message, it should produce no output, i.e., its output should be $\perp$. This means its denotation should be strict in the component c^+. To make a function f strict in a component $j \in I$, we use the continuous function $\mathrm{strict}_j : [\prod_{i \in I} A_i \to B] \to [\prod_{i \in I} A_i \to B]$:

$$\mathrm{strict}_j(f)\left((a_i)_{i \in I}\right) = \begin{cases} \perp_B & \text{if } a_j = \perp_{A_j} \\ f\left((a_i)_{i \in I}\right) & \text{otherwise.} \end{cases} \tag{8}$$

We use this to define the interpretation of $(\mathbf{1}L)$:

$$\begin{aligned}&[\![\Psi\ ;\ \Delta, c : \mathbf{1} \vdash \mathsf{wait}\ c; P :: a : A]\!]u : [\![\Delta, c : \mathbf{1}]\!]^+ \times [\![a : A]\!]^- \to [\![\Delta, c : \mathbf{1}]\!]^- \times [\![a : A]\!]^+\\ &[\![\Psi\ ;\ \Delta, c : \mathbf{1} \vdash \mathsf{wait}\ c; P :: a : A]\!]u = \mathsf{strict}_{c^+}\left(\lambda(\delta^+, c^+, a^-).(\delta^-, \bot, a^+)\right))\end{aligned} \tag{9}$$

where $(\delta^-, a^+) = [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u(\delta^+, a^-)$.

The semantic clauses (8) and (9) are instances of general principles in our semantics. The absence of communication corresponds to the bottom element $\bot$. When a process waits for input on a channel c it uses (it provides), its denotation is strict in c^+ (resp., c^-). When a process sends on a channel c it provides (it uses), its denotation is not strict in c^- (resp., c^+).

Proposition 3.3. *For all $\Psi\ ;\ \Delta \vdash P :: a : A$, we have the equivalence $P \equiv c \leftarrow \mathsf{close}\ c;\ (\mathsf{wait}\ c; P)$.*

Proof. Let $u \in [\![\Psi]\!]$ be arbitrary and let

$$\begin{aligned}g &= \lambda(\delta^+, c^+, a^-) \in [\![\Delta, c : \mathbf{1}]\!]^+ \times [\![a : A]\!]^-.(\delta^-, \bot, a^+)\\ &\qquad \text{where } (\delta^-, a^+) = [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u(\delta^+, a^-),\\ f &= [\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u \times \mathsf{strict}_{c^+}\, g.\end{aligned}$$

So $[\![\Psi\ ;\ \Delta \vdash c \leftarrow \mathsf{close}\ c;\ (\mathsf{wait}\ c; P) :: a : A]\!]u = f^{\ddagger}_{c^- \times c^+}$. The function $\pi_{c^+} \circ [\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u$ is not strict. By proposition 4.6 this implies

$$f^{\ddagger}_{c^- \times c^+} = \left([\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u \times g\right)^{\ddagger}_{c^- \times c^+}. \tag{10}$$

The denotation of P does not depend on c and the product in eq. (10) can be written as

$$[\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u \times g = \left(\lambda(c^-, c^+).(c^+ : *, c^- : \bot)\right) \times [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u.$$

By proposition 4.3 this implies $\left([\![\Psi\ ;\ \cdot \vdash \mathsf{close}\ c :: c : \mathbf{1}]\!]u \times g\right)^{\ddagger}_{c^- \times c^+} = [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u$. We conclude $[\![\Psi\ ;\ \Delta \vdash c \leftarrow \mathsf{close}\ c;\ (\mathsf{wait}\ c; P) :: a : A]\!]u = [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u$ by transitivity. □

3.3 Shifts in polarity

Shifts in polarity can be used to enforce synchronous communication in asynchronous settings [Pfenning and Griffith 2015]. For example, to synchronize on a channel $a : A$ with A negative, we "downshift" it to the positive type $\downarrow A$. The provider of a signals that it is ready to receive messages on a by sending the "shift" message over a. The client cannot send messages to the provider until it has received the shift message. This is captured by the rules:

$$\frac{\Xi \vdash A\ \mathsf{type}^-_{\mathsf{s}}}{\Xi \vdash \downarrow A\ \mathsf{ctype}^+_{\mathsf{s}}}\ (\mathrm{C}\downarrow) \qquad \frac{\Psi\ ;\ \Delta \vdash P :: a : A}{\Psi\ ;\ \Delta \vdash \mathsf{send}\ a\ \mathsf{shift}; P :: a : \downarrow A}\ (\downarrow R) \qquad \frac{\Psi\ ;\ \Delta, a : A \vdash P :: c : C}{\Psi\ ;\ \Delta, a : \downarrow A \vdash \mathsf{shift} \leftarrow \mathsf{recv}\ a; P :: c : C}\ (\downarrow L)$$

The elements of $[\![\downarrow A]\!]$ capture either no communication (the element $\bot_{[\![\downarrow A]\!]}$), or a shift message potentially followed by a communication of type A. We use the element $a \in [\![A]\!]$ to capture the second case. However, we must distinguish between the no communication and a shift message followed by no communication of type A (the element $\bot_{[\![A]\!]}$). We accomplish this by adjoining a new bottom element $\bot_{[\![\downarrow A]\!]}$ to $[\![A]\!]$, i.e., by *lifting* $[\![A]\!]$.

The **lifting functors** $(-)_\bot : \omega\text{-}\mathbf{aBC} \to \omega\text{-}\mathbf{aBC}$ and $(-)_\bot : \omega\text{-}\mathbf{aBC}_{\bot!} \to \omega\text{-}\mathbf{aBC}_{\bot!}$ are respectively left-adjoint to the identity functors $\mathsf{id}_{\omega\text{-}\mathbf{aBC}}$ and $\mathsf{id}_{\omega\text{-}\mathbf{aBC}_{\bot!}}$. The domain $D_\bot$ is obtained by adjoining a new bottom element to D. The morphism $f_\bot : D_\bot \to E_\bot$ is given by $f_\bot(\bot_{D_\bot}) = \bot_{E_\bot}$ and $f_\bot(d) = f(d)$ for $d \in D$. The units $\mathsf{id} \to (-)_\bot$ are respectively called up and $\mathsf{up}_{\bot!}$. The counits $(-)_\bot \to \mathsf{id}$ are equal and are called down. Write $d_\bot$ for $\mathsf{up}_D(d) \in D_\bot$. The lifting functor on $\omega\text{-}\mathbf{aBC}_{\bot!}$ restricts to a functor $\mathbf{M} \to \mathbf{M}$. The components of $\mathsf{up}_{\bot!}$ are not meet-homomorphisms in general, but the components of down are. Given a functor F, we abbreviate the composition $(-)_\bot \circ F$ as $F_\bot$.

Downshifting A introduces only *positive* communication: the shift message travels from left to right. We capture this by lifting the positive aspect. Because no negative information was added, the negative aspect $\downarrow A$ is the same as the negative aspect of A. The relating natural transformations for proposition 2.1 are determined by $\pi^-_{\downarrow A} = \mathsf{down} * \pi^-_A$ and $\pi^+_{\downarrow A} = \left(\pi^+_A\right)_\perp$.

$$[\![\Xi \vdash \downarrow A\ \mathsf{ctype_s}]\!] = [\![\Xi \vdash A\ \mathsf{type_s}]\!]_\perp, \tag{11}$$

$$[\![\Xi \vdash \downarrow A\ \mathsf{ctype_s}]\!]^- = [\![\Xi \vdash A\ \mathsf{type_s}]\!]^-, \tag{12}$$

$$[\![\Xi \vdash \downarrow A\ \mathsf{ctype_s}]\!]^+ = [\![\Xi \vdash A\ \mathsf{type_s}]\!]^+_\perp. \tag{13}$$

The semantic clauses (11) and (13) (and (4) and (5) for $\mathbf{1}$) are instances of another general principle in our semantics. When $A = F(A_1, \ldots, A_n)$ is a type of polarity p for some type former F and subphrases A_i of polarity p, then $[\![A]\!]^p$ is defined using the $[\![A_i]\!]^p$ in the same way $[\![A]\!]$ is defined using the $[\![A_i]\!]$.

In our asynchronous setting, the shift message is always sent. This corresponds to lifting the output of P on the a^+ component. We interpret $(\downarrow R)$ as:

$$\begin{aligned}&[\![\Psi\ ;\ \Delta \vdash \mathsf{send}\ a\ \mathsf{shift}; P :: a : \downarrow A]\!]u : [\![\Delta]\!]^+ \times \left(a^- : [\![A]\!]^-\right) \to [\![\Delta]\!]^- \times \left(a^+ : [\![A]\!]^+_\perp\right)\\ &[\![\Psi\ ;\ \Delta \vdash \mathsf{send}\ a\ \mathsf{shift}; P :: a : \downarrow A]\!]u = \left(\mathsf{id} \times \left(a^+ : \mathsf{up}\right)\right) \circ [\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u.\end{aligned} \tag{14}$$

The client waits until it receives the shift message on a^+. This corresponds to being strict in the a^+ component. We lower $[\![A]\!]^+_\perp$ to $[\![A]\!]^+$ to extract the positive communication expected by P.

$$\begin{aligned}&[\![\Psi\ ;\ \Delta, a : \downarrow A \vdash \mathsf{shift} \leftarrow \mathsf{recv}\ a; P :: c : C]\!]u :\\ &\qquad : [\![\Delta]\!]^+ \times \left(a^+ : [\![A]\!]^+_\perp\right) \times [\![c : C]\!]^- \to [\![\Delta]\!]^- \times \left(a^- : [\![A]\!]^-\right) \times [\![c : C]\!]^+\\ &[\![\Psi\ ;\ \Delta, a : \downarrow A \vdash \mathsf{shift} \leftarrow \mathsf{recv}\ a; P :: c : C]\!]u\\ &= \mathsf{strict}_{a^+}\left([\![\Psi\ ;\ \Delta, a : A \vdash P :: c : C]\!]u \circ \left(\mathsf{id} \times \left(a^+ : \mathsf{down}\right)\right)\right).\end{aligned} \tag{15}$$

A useful property of the units and counits for the lifting functors is that they form section-retraction pairs. A pair (s, r) of morphisms $s : A \to B$ and $r : B \to A$ forms a **section-retraction pair** if $r \circ s = \mathsf{id}_A$. We say that a pair (σ, ρ) of natural transformations $\sigma : F \to G$ and $\rho : G \to F$ form a section-retraction pair if each pair of components (σ_C, ρ_C) is a section-retraction pair. In this case, the natural transformations $(\mathsf{up}, \mathsf{down})$ and $\left(\mathsf{up}_{\perp!}, \mathsf{down}\right)$ are section-retraction pairs.

Proposition 3.4. *For all* $\Psi\ ;\ \Delta_1 \vdash P :: a : A$ *and* $\Psi\ ;\ \Delta_2, a : A \vdash Q :: c : C$,

$$a \leftarrow P;\ Q \equiv a \leftarrow (\mathsf{send}\ a\ \mathsf{shift}; P);\ (\mathsf{shift} \leftarrow \mathsf{recv}\ a; Q).$$

Proof. We abbreviate typing judgments by their processes. The function $\pi_{a^+} \circ [\![\mathsf{send}\ a\ \mathsf{shift}; P]\!]u$ is not strict because of the post-composition with up in the a^+ component. By proposition 4.6, this implies we can drop the strict_{a^+} when computing the denotation of the cut processes:

$$\begin{aligned}&[\![\Psi\ ;\ \Delta_1, \Delta_2 \vdash a \leftarrow (\mathsf{send}\ a\ \mathsf{shift}; P)\ ;\ (\mathsf{shift} \leftarrow \mathsf{recv}\ a; Q) :: c : C]\!]u\\ &= \left([\![\mathsf{send}\ a\ \mathsf{shift}; P]\!]u \times [\![\mathsf{shift} \leftarrow \mathsf{recv}\ a; Q]\!]u\right)^{\ddagger}_{a^- \times a^+}\\ &= \left(\left(\left(\mathsf{id} \times \left(a^+ : \mathsf{up}\right)\right) \circ [\![P]\!]u\right) \times \mathsf{strict}_{a^+}\left([\![Q]\!]u \circ \left(\mathsf{id} \times \left(a^+ : \mathsf{down}\right)\right)\right)\right)^{\ddagger}_{a^- \times a^+}\\ &= \left(\left(\left(\mathsf{id} \times \left(a^+ : \mathsf{up}\right)\right) \circ [\![P]\!]u\right) \times \left([\![Q]\!]u \circ \left(\mathsf{id} \times \left(a^+ : \mathsf{down}\right)\right)\right)\right)^{\ddagger}_{a^- \times a^+}\\ &= \left(\left(\mathsf{id} \times \left(a^+ : \mathsf{up}\right)\right) \circ \left([\![P]\!]u \times [\![Q]\!]u\right) \circ \left(\mathsf{id} \times \left(a^+ : \mathsf{down}\right)\right)\right)^{\ddagger}_{a^- \times a^+}.\end{aligned}$$

Because (up, down) is a continuous section-retraction pair and down is strict, by proposition 4.5

$$\begin{aligned}
&[\![\Psi \mathrel{;} \Delta_1, \Delta_2 \vdash a \leftarrow (\mathsf{send}\ a\ \mathsf{shift}; P)\mathrel{;}\ (\mathsf{shift} \leftarrow \mathsf{recv}\ a; Q) :: c : C]\!]u \\
&= \left([\![P]\!]u \times [\![Q]\!]u\right)^{\ddagger}_{a^- \times a^+} \\
&= [\![\Psi \mathrel{;} \Delta_1, \Delta_2 \vdash a \leftarrow P;\ Q :: c : C]\!]u.
\end{aligned}$$ □

Dually, we can "upshift" a positive type A to the negative type $\uparrow A$. This is captured by the following three rules. Their semantic development is symmetric in polarity to the one for $\downarrow A$.

$$\frac{\Xi \vdash A\ \mathsf{type}^+_{\mathsf{s}}}{\Xi \vdash \uparrow A\ \mathsf{ctype}^-_{\mathsf{s}}}\ (\mathrm{C}\uparrow) \qquad \frac{\Psi \mathrel{;} \Delta \vdash P :: a :}{\Psi \mathrel{;} \Delta \vdash \mathsf{shift} \leftarrow \mathsf{recv}\ a; P :: a : \uparrow A}\ (\uparrow R) \qquad \frac{\Psi \mathrel{;} \Delta, a : A \vdash P :: c : C}{\Psi \mathrel{;} \Delta, a : \uparrow A \vdash \mathsf{send}\ a\ \mathsf{shift}; P :: c : C}\ (\uparrow L)$$

3.4 Making choices

A provider can choose at run time the type of service it provides. When it chooses between session types $\{A_l\}_{l \in L}$ (with L finite), it provides the internal choice type $\oplus\{l : A_l\}_{l \in L}$. A process chooses to provide the service A_k by sending the label k over the channel. The process syntax is $a.k; P$, where a is the provided channel, k is the label, and P is the continuation process providing the service of type A_k. The client case $a\ \{l \Rightarrow P_l\}_{l \in L}$ blocks until it receives a choice label k on a. It then continues as the process P_k.

$$\frac{\Psi \mathrel{;} \Delta \vdash P :: a : A_k \quad (k \in L)}{\Psi \mathrel{;} \Delta \vdash a.k; P :: a : \oplus\{l : A_l\}_{l \in L}}\ (\oplus R_k) \qquad \frac{\Psi \mathrel{;} \Delta, a : A_l \vdash P_l :: c : C \quad (\forall l \in L)}{\Psi \mathrel{;} \Delta, a : \oplus\{l : A_l\}_{l \in L} \vdash \mathsf{case}\ a\ \{l_l \Rightarrow P_l\}_{i \in I} :: c : C}\ (\oplus L)$$

$$\frac{\Xi \vdash A_l\ \mathsf{type}^+_{\mathsf{s}} \quad (\forall l \in L)}{\Xi \vdash \oplus\{l : A_l\}_{l \in L}\ \mathsf{ctype}^+_{\mathsf{s}}}\ (\mathrm{C}\oplus)$$

Sending a label k and continuing with communications of type A_k corresponds to tagging a communication $a_k \in [\![A_k]\!]$ with the label k. These tagged communications (k, a_k) are the elements of the disjoint union $\biguplus_{l \in L} [\![A_l]\!]$. To account for the potential lack of communication, we lift this disjoint union to adjoin a new bottom element. This lifted disjoint union is isomorphic to the coalesced sum $\bigoplus_{l \in L} [\![A_l]\!]_\perp$. We identify their elements using this isomorphism. Explicitly, the elements are $\perp$ and $(k, a_k)_\perp$ for $(k, a_k) \in \biguplus_{l \in L} [\![A_l]\!]$. Coalesced sums are coproducts in ω-$\mathbf{aBC}_{\perp!}$. We define the interpretations using coalesced sums to make this structure evident. The provider sends the label on the positive aspect of the channel, justifying eq. (18). The client does not know a priori which branch it will take: it must be ready to send negative information for each possible branch. This justifies eq. (17).

$$[\![\Xi \vdash \oplus\{l : A_l\}_{l \in L}\ \mathsf{ctype}_{\mathsf{s}}]\!] = \bigoplus_{l \in L} [\![\Xi \vdash A_l\ \mathsf{type}_{\mathsf{s}}]\!]_\perp, \tag{16}$$

$$[\![\Xi \vdash \oplus\{l : A_l\}_{l \in L}\ \mathsf{ctype}_{\mathsf{s}}]\!]^- = \prod_{l \in L} [\![\Xi \vdash A_l\ \mathsf{type}_{\mathsf{s}}]\!]^-, \tag{17}$$

$$[\![\Xi \vdash \oplus\{l : A_l\}_{l \in L}\ \mathsf{ctype}_{\mathsf{s}}]\!]^+ = \bigoplus_{l \in L} [\![\Xi \vdash A_i\ \mathsf{type}_{\mathsf{s}}]\!]^+_\perp. \tag{18}$$

The category ω-$\mathbf{aBC}_{\perp!}$ has zero morphisms and we use matrix notation for morphisms from coproducts to products [Riehl 2016, pp. 82f.]. The relating natural transformations for proposition 2.1 are determined by $\pi^-_{\oplus\{l:A_l\}_{l \in L}} = \mathrm{diag}\left(\mathsf{down} * \pi^-_{A_l}\right)_{l \in L}$ and $\pi^+_{\oplus\{l:A_l\}_{l \in L}} = \bigoplus_{l \in L} \left(\pi^+_{A_l}\right)_\perp$. Their components are meet-homomorphisms. Explicitly, $\pi^-_{\oplus\{l:A_l\}_{l \in L}}(k, a_k) = (k : (\mathsf{down} * \pi_{A_k})(a_k), l \neq k : \perp)_{l \in L}$.

In the interpretation of $(\oplus R_k)$, the provider extracts from a^- the negative information a_k^- required by the continuation process P. Its output on a^+ is the label k, followed by the output of P on a^+:

$$\begin{aligned}
&[\![\Psi\ ;\ \Delta \vdash a.k; P :: a : \oplus\{l : A_l\}_{l\in L}]\!]u : [\![\Delta]\!]^+ \times \left(a^- : \prod_{l\in L} [\![A_l]\!]^-\right) \to [\![\Delta]\!]^- \times \left(a^+ : \bigoplus_{l\in L} [\![A_l]\!]^+_\perp\right) \\
&[\![\Psi\ ;\ \Delta \vdash a.k; P :: a : \oplus\{l : A_l\}_{l\in L}]\!]u \left(\delta^+, \left(a_l^-\right)_{l\in L}\right) = \left(\delta^-, \left(k, a_k^+\right)_\perp\right),
\end{aligned} \tag{19}$$

where $[\![\Psi\ ;\ \Delta \vdash P :: a : A_k]\!]u \left(\delta^+, a_k^-\right) = \left(\delta^-, a_k^+\right)$.

The client case $a\ \{l \Rightarrow P_l\}_{l\in L}$ waits until it receives a label on a^+, i.e., it is strict in the a^+ component. Given a communication $(k, a_k^+)_\perp$ on a^+, it processes a_k^+ using the process P_k. The process P_k produces a negative communication a_k^-. To transmit this back to the provider, we place it in the k-component of the product sent on a^-. The client produces no communications for the other possible branches, so it fills those other components with $\perp$.

$$\begin{aligned}
&[\![\Psi\ ;\ \Delta, a : \oplus\{l : A_l\}_{l\in L} \vdash \mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L} :: c : C]\!]u : \\
&\qquad : [\![\Delta]\!]^+ \times \left(a^+ : \bigoplus_{l\in L} [\![A_l]\!]^+_\perp\right) \times [\![c : C]\!]^- \to [\![\Delta]\!]^- \times \left(a^- : \prod_{l\in L} [\![A_l]\!]^-\right) \times [\![c : C]\!]^+ \\
&[\![\Psi\ ;\ \Delta, a : \oplus\{l : A_l\}_{l\in L} \vdash \mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L} :: c : C]\!]u \\
&= \mathrm{strict}_{a^+} \left(\lambda \left(\delta^+, a^+ : \left(k, a_k^+\right)_\perp, c^-\right) . \left(\delta^-, a^- : \left(k : a_k^-, l \neq k : \perp\right)_{l\in L}, c^+\right)\right)
\end{aligned} \tag{20}$$

where $(\delta^-, a_k^-, c^+) = [\![\Psi\ ;\ \Delta, a : A_k \vdash P_k :: c : C]\!]u(\delta^+, a_k^+, c^-)$. We abuse notation to pattern match on the component a^+. By strictness, we know that it will be an element of the form $\left(k, a_k^+\right)_\perp$.

Proposition 3.5. *Let L be a finite set and $k \in L$. Consider processes $\Psi\ ;\ \Delta_1 \vdash P :: a : A_k$ and $\Psi\ ;\ \Delta_2, a : A_l \vdash Q_l :: c : C$ for all $l \in L$, then $a \leftarrow P;\ Q_k \equiv a \leftarrow (a.k; P);\ (\mathsf{case}\ a\ \{l \Rightarrow Q_l\}_{l\in L})$.*

Proof. We use the Knaster-Tarski formulation (66) of the double-dagger operation. Let $u \in [\![\Psi]\!]$ and $(\delta_1^+, \delta_2^+, c^-) \in [\![\Delta_1, \Delta_2]\!]^+ \times [\![c : C]\!]^-$ be arbitrary. We abbreviate judgments by their processes and calculate that

$$\begin{aligned}
&[\![a \leftarrow P;\ Q_k]\!]u(\delta_1^+, \delta_2^+, c^-) = l^\ddagger_{b^-\times b^+}(\delta_1^+, \delta_2^+, c^-) = \pi_{\Delta_1^-, \Delta_2^-, c^+} \left(\bigsqcap L\right), \\
&[\![a \leftarrow (a.k; P);\ (\mathsf{case}\ a\ \{l \Rightarrow Q_l\}_{l\in L})]\!]u(\delta_1^+, \delta_2^+, c^-) = r^\ddagger_{b^-\times b^+}(\delta_1^+, \delta_2^+, c^-) = \pi_{\Delta_1^-, \Delta_2^-, c^+} \left(\bigsqcap R\right)
\end{aligned}$$

where

$$\begin{aligned}
&l : [\![\Delta_1, \Delta_2, a : A_k]\!]^+ \times [\![a : A_k, c : C]\!]^- \to [\![\Delta_1, \Delta_2, a : A_k]\!]^- \times [\![a : A_k, c : C]\!]^+ \\
&l = [\![\Psi\ ;\ \Delta_1 \vdash P :: a : A_k]\!]u \times [\![\Psi\ ;\ \Delta_2, a : A_k \vdash Q_k :: c : C]\!]u, \\
&L = \left\{ \left(\delta_1^-, \delta_2^-, a^-, a^+, c^+\right) \in [\![\Delta_1, \Delta_2, a : A_k]\!]^- \times [\![a : A_k, c : C]\!]^+ \mid \right. \\
&\qquad \left. \mid l\left(\delta_1^+, \delta_2^+, a^+, a^-, c^-\right) \sqsubseteq \left(\delta_1^-, \delta_2^-, a^-, a^+, c^+\right) \right\}, \\
&r : [\![\Delta_1, \Delta_2, a : \oplus\{l : A_l\}_{l\in L}]\!]^+ \times [\![a : \oplus\{l : A_l\}_{l\in L}, c : C]\!]^- \to \\
&\qquad \to [\![\Delta_1, \Delta_2, a : \oplus\{l : A_l\}_{l\in L}]\!]^- \times [\![a : \oplus\{l : A_l\}_{l\in L}, c : C]\!]^+ \\
&r = [\![\Psi\ ;\ \Delta_1 \vdash a.k; P :: a : \oplus\{l : A_l\}_{l\in L}]\!]u \times \\
&\qquad \times [\![\Psi\ ;\ \Delta_2, a : \oplus\{l : A_l\}_{l\in L} \vdash \mathsf{case}\ a\ \{l \Rightarrow Q_l\}_{l\in L} :: c : C]\!]u,
\end{aligned}$$

$$R = \left\{ \left(\delta_1^-, \delta_2^-, a^-, a^+, c^+\right) \in [\![\Delta_1, \Delta_2, a : \oplus\{l : A_l\}_{l\in L}]\!]^- \times [\![a : \oplus\{l : A_l\}_{l\in L}, c : C]\!]^+ \mid \right.$$
$$\left. \mid r\left(\delta_1^+, \delta_2^+, a^+, a^-, c^-\right) \sqsubseteq \left(\delta_1^-, \delta_2^-, a^-, a^+, c^+\right) \right\}.$$

Infima of products are computed component-wise, so to show

$$[\![a \leftarrow P;\ Q_k]\!]u(\delta_1^+, \delta_2^+, c^-) = [\![a \leftarrow (a.k; P);\ (\mathsf{case}\ a\ \{l \Rightarrow Q_l\}_{l\in L})]\!]u(\delta_1^+, \delta_2^+, c^-)$$

it is sufficient to show that $\pi_{\Delta_1^-,\Delta_2^-,c^+}L = \pi_{\Delta_1^-,\Delta_2^-,c^+}R$.

We first show $\pi_{\Delta_1^-,\Delta_2^-,c^+}L \supseteq \pi_{\Delta_1^-,\Delta_2^-,c^+}R$. Let $(\delta_1^-, \delta_2^-, (a_l^-)_{l\in L}, a^+, c^+) \in R$ be arbitrary and let $[\![P]\!]u(\delta_1^+, a_k^-) = (\delta_P^-, a_P^+)$. The case $a^+ = \bot$ is impossible. Indeed, when $a^+ = \bot$, strictness gives $[\![\mathsf{case}\ a\ \{l \Rightarrow Q_l\}_{l\in L}]\!]u(\delta_2^+, \bot, c^-) = \bot$. This implies

$$r(\delta_1^-, \delta_2^-, a^+ : \bot, (a_l^-)_{l\in L}, c^+) = (\Delta_1^- : \delta_P^-, \Delta_2^- : \bot, a^- : \bot, a^+ : (k, a_P^+)_\bot, c^+ : \bot),$$

which by definition of R implies the contradiction $(k, a_P^+)_\bot \sqsubseteq \bot$. So $a^+ = (l, a_l^+)_\bot$ for some $l \in L$ and $a_l^+ \in [\![A_l]\!]^+$. The definitions of r and R imply $(k, a_P^+)_\bot \sqsubseteq (l, a_l^+)_\bot$, so we have $l = k$. We show that $(\delta_1^-, \delta_2^-, a_k^-, a_k^+, c^+) \in L$. Let $[\![Q_k]\!]u(\delta_2^+, a_k^+, c^-) = (\delta_Q^-, a_Q^-, c_Q^+)$. We calculate

$$l(\delta_1^+, \delta_2^+, a_k^+, a_k^-, c^-) = (\Delta_1^- : \delta_P^-, \Delta_2^- : \delta_Q^-, a^- : a_Q^-, a^+ : a_P^+, c^+ : c_Q^+), \tag{21}$$

$$\begin{aligned} &r(\delta_1^-, \delta_2^-, (k, a_k^+)_\bot, (a_l^-)_{l\in L}, c^+) \\ &= (\Delta_1^- : \delta_P^-, \Delta_2^- : \delta_Q^-, a^- : (k : a_Q^-, l \neq k : \bot), a^+ : (k, a_P^+)_\bot, c^+ : c_Q^+). \end{aligned} \tag{22}$$

By definition of R, eq. (22) implies

$$(\delta_P^-, \delta_Q^-, (k : a_Q^-, l \neq k : \bot), (k, a_P^+)_\bot, c_Q^+) \sqsubseteq (\delta_1^-, \delta_2^-, (a_l^-)_{l\in L}, (k, a_k^+)_\bot, c^+). \tag{23}$$

It immediately follows from (23) that

$$(\delta_P^-, \delta_Q^-, a_Q^-, a_P^+, c_Q^+) \sqsubseteq (\delta_1^-, \delta_2^-, a_k^-, a_k^+, c^+),$$

i.e., that $(\delta_1^-, \delta_2^-, a_k^-, a_k^+, c^+) \in L$. We deduce $\pi_{\Delta_1^-,\Delta_2^-,c^+}L \supseteq \pi_{\Delta_1^-,\Delta_2^-,c^+}R$.

To show $\pi_{\Delta_1^-,\Delta_2^-,c^+}L \subseteq \pi_{\Delta_1^-,\Delta_2^-,c^+}R$, let $(\delta_1^-, \delta_2^-, a_k^-, a_k^+, c^+) \in L$ be arbitrary. A similar argument gives that $(\delta_1^-, \delta_2^-, (k : a_k^-, l \neq k : \bot), (k, a_k^+)_\bot, c^+) \in R$. □

Dually, a provider can accept external choices. The semantic development for external choices $\&\{l : A_l\}_{l\in L}$ is dual to the one for internal choices.

$$\frac{\Psi\ ;\ \Delta \vdash P_l :: a : A_l \quad (\forall l \in L)}{\Psi\ ;\ \Delta \vdash \mathsf{case}\ a\ \{l \Rightarrow P_l\}_{l\in L} :: a : \&\{l : A_l\}_{l\in L}}\ (\&R) \qquad \frac{\Psi\ ;\ \Delta, a : A_k \vdash P :: c : C \quad (k \in L)}{\Psi\ ;\ \Delta, a : \&\{l : A_l\}_{l\in L} \vdash a.k; P :: c : C}\ (\&L_k)$$

$$\frac{\Xi \vdash A_l\ \mathsf{type}_\mathsf{s}^- \quad (\forall l \in L)}{\Xi \vdash \&\{l : A_l\}_{l\in L}\ \mathsf{ctype}_\mathsf{s}^-}\ (\mathrm{C}\&)$$

3.5 Channel transmission

The provider send $b\ a;\ P$ sends a channel a over the channel b before continuing as P. When the client $a \leftarrow \mathsf{recv}\ b;\ P$ receives a channel over b, it binds it to the name a and continues as P. The type $A \otimes B$ describes a service that sends a channel of type A and becomes a channel of type B. This is captured by the rules:

$$\frac{\Psi\ ;\ \Delta \vdash P :: b : B}{\Psi\ ;\ \Delta, a : A \vdash \mathsf{send}\ b\ a;\ P :: b : A \otimes B}\ (\otimes R^*) \qquad \frac{\Psi\ ;\ \Delta, a : A, b : B \vdash P :: c : C}{\Psi\ ;\ \Delta, b : A \otimes B \vdash a \leftarrow \mathsf{recv}\ b;\ P :: c : C}\ (\otimes L)$$

$$\frac{\Xi \vdash A\ \mathsf{type}_\mathsf{s}^+ \quad \Xi \vdash B\ \mathsf{type}_\mathsf{s}^+}{\Xi \vdash A \otimes B\ \mathsf{ctype}_\mathsf{s}^+}\ (\mathrm{C}\otimes)$$

We cannot directly observe a channel, only the messages that are sent over it. For this reason, we treat communications of type $A \otimes B$ as a pair of communications: one for the sent channel and one

for the continuation channel. This is analogous to the denotation of $A \otimes B$ given by Atkey [2017]. We account for the potential absence of communication by lifting.

$$[\![\Xi \vdash A \otimes B\ \mathsf{ctype_s}]\!] = \left([\![\Xi \vdash A\ \mathsf{type_s}]\!] \times [\![\Xi \vdash B\ \mathsf{type_s}]\!]\right)_\perp, \tag{24}$$

$$[\![\Xi \vdash A \otimes B\ \mathsf{ctype_s}]\!]^- = [\![\Xi \vdash A\ \mathsf{type_s}]\!]^- \times [\![\Xi \vdash B\ \mathsf{type_s}]\!]^-, \tag{25}$$

$$[\![\Xi \vdash A \otimes B\ \mathsf{ctype_s}]\!]^+ = \left([\![\Xi \vdash A\ \mathsf{type_s}]\!]^+ \times [\![\Xi \vdash B\ \mathsf{type_s}]\!]^+\right)_\perp. \tag{26}$$

The relating natural transformations for proposition 2.1 are determined by $\pi^-_{A\otimes B} = \mathsf{down}*\left(\pi^-_A \times \pi^-_B\right)$ and $\pi^+_{A\otimes B} = \left(\pi^+_A \times \pi^+_B\right)_\perp$.

To send the channel a over b, the provider send $b\ a;\ P$ must relay the positive communication from a^+ to the $[\![A]\!]^+$-component of $[\![b : A \otimes B]\!]^+$. It must also relay the negative information on the $[\![A]\!]^-$-component of $[\![b : A \otimes B]\!]^-$ to a^-. The process P handles communication for the channels Δ and the B-component of b. This gives the semantic clause:

$$\begin{aligned}
&[\![\Psi\ ;\ \Delta, a : A \vdash \mathsf{send}\ b\ a;\ P :: b : A \otimes B]\!]u : \\
&\qquad : [\![\Delta, a : A]\!]^+ \times \left(b^- : [\![A]\!]^- \times [\![B]\!]^-\right) \to [\![\Delta, a : A]\!]^- \times \left(b^+ : \left([\![A]\!]^+ \times [\![B]\!]^+\right)_\perp\right) \\
&[\![\Psi\ ;\ \Delta, a : A \vdash \mathsf{send}\ b\ a;\ P :: b : A \otimes B]\!]u(\delta^+, a^+, (b^-_a, b^-_b)) = \left(\delta^-, b^-_a, \left(a^+, b^+_b\right)_\perp\right)
\end{aligned} \tag{27}$$

where $[\![\Psi\ ;\ \Delta \vdash P :: b : B]\!]u(\delta^+, b^-_b) = (\delta^-, b^+_b)$.

The client $a \leftarrow \mathsf{recv}\ b;\ Q$ waits until it receives a channel on b, i.e., it is strict in the b^+ component. When it receives a positive communication (b^+_a, b^+_b) on $[\![b : A \otimes B]\!]^+$, it must unpack it into the two positive channels $[\![a : A, b : B]\!]^+$ expected by Q. It must then repack the negative information $[\![a : A, b : B]\!]^-$ produced by Q and relay it over $[\![b : A \otimes B]\!]^-$. This corresponds to:

$$\begin{aligned}
&[\![\Psi\ ;\ \Delta, b : A \otimes B \vdash a \leftarrow \mathsf{recv}\ b;\ Q :: c : C]\!]u : \\
&\qquad : [\![\Delta]\!]^+ \times \left(b^+ : \left([\![A]\!]^+ \times [\![B]\!]^+\right)_\perp\right) \times [\![c : C]\!]^- \to [\![\Delta]\!]^- \times \left(b^- : [\![A]\!]^- \times [\![B]\!]^-\right) \times [\![c : C]\!]^+ \\
&[\![\Psi\ ;\ \Delta, b : A \otimes B \vdash a \leftarrow \mathsf{recv}\ b;\ Q :: c : C]\!]u(\delta^+, b^+, c^-) \\
&= \mathsf{strict}_{b^+}\left(\lambda(\delta^+, b^+ : (b^+_a, b^+_b)_\perp, c^-).(\delta^-, (a^-, b^-), c^+)\right)
\end{aligned} \tag{28}$$

where $[\![\Psi\ ;\ \Delta, a : A, b : B \vdash Q :: c : C]\!]u(\delta^+, b^+_a, b^+_b, c^-) = (\delta^-, a^-, b^-, c^+)$. As in eq. (20), we abuse notation to pattern match on the component b^+.

Proposition 3.6. *Consider processes $\Psi\ ;\ \Delta_1 \vdash P :: b : B$ and $\Psi\ ;\ \Delta_2, a : A, b : B \vdash Q :: c : C$. The rules ($\otimes R^*$) and ($\otimes L$) respectively form processes $\Psi\ ;\ \Delta_1, a : A \vdash \mathsf{send}\ b\ a;\ P :: b : A \otimes B$ and $\Psi\ ;\ \Delta_2, b : A \otimes B \vdash a \leftarrow \mathsf{recv}\ b;\ Q :: c : C$. Then $b \leftarrow P;\ Q \equiv b \leftarrow (\mathsf{send}\ b\ a;\ P)\,;\ (a \leftarrow \mathsf{recv}\ b;\ Q)$.*

Proof. We use the Knaster-Tarski formulation (66) of the double-dagger operation. Let $u \in [\![\Psi]\!]$ and $(\delta^+_1, \delta^+_2, a^+, c^-) \in [\![\Delta_1, \Delta_2, a : A]\!]^+ \times [\![c : C]\!]^-$ be arbitrary. We abbreviate judgments by their processes and calculate that $[\![b \leftarrow (\mathsf{send}\ b\ a;\ P)\,;\ (a \leftarrow \mathsf{recv}\ b;\ Q)]\!]u(\delta^+_1, \delta^+_2, a^+, c^-) = \pi_{\Delta^-_1, \Delta^-_2, a^-, c^+}\left(\bigcap L\right)$ and $[\![b \leftarrow P;\ Q]\!]u(\delta^+_1, \delta^+_2, a^+, c^-) = \pi_{\Delta^-_1, \Delta^-_2, a^-, c^+}\left(\bigcap R\right)$ where

$$\begin{aligned}
l &= [\![\Psi\ ;\ \Delta_1, a : A \vdash \mathsf{send}\ b\ a;\ P :: b : A \otimes B]\!]u \times [\![\Psi\ ;\ \Delta_2, b : A \otimes B \vdash a \leftarrow \mathsf{recv}\ b;\ Q :: c : C]\!]u, \\
L &= \left\{\left(\delta^-_1, \delta^-_2, a^-, b^-, b^+, c^+\right) \in [\![\Delta_1, \Delta_2, a : A, b : A \otimes B]\!]^- \times [\![b : A \otimes B, c : C]\!]^+ \mid \right. \\
&\qquad \left. \mid l(\delta^+_1, \delta^+_2, a^+, b^+, b^-, c^-) \sqsubseteq (\delta^-_1, \delta^-_2, a^-, b^-, b^+, c^+)\right\}, \\
r &= [\![\Psi\ ;\ \Delta_1 \vdash P :: b : B]\!]u \times [\![\Psi\ ;\ \Delta_2, a : A, b : B \vdash Q :: c : C]\!]u, \\
R &= \left\{\left(\delta^-_1, \delta^-_2, a^-, b^-, b^+, c^+\right) \in [\![\Delta_1, \Delta_2, a : A, b : B]\!]^- \times [\![b : B, c : C]\!]^+ \mid \right. \\
&\qquad \left. \mid r(\delta^+_1, \delta^+_2, a^+, b^+, b^-, c^-) \sqsubseteq (\delta^-_1, \delta^-_2, a^-, b^-, b^+, c^+)\right\}.
\end{aligned}$$

Infima of products are computed component-wise, so to show

$$[\![b \leftarrow (\mathsf{send}\ b\ a;\ P);\ (a \leftarrow \mathsf{recv}\ b;\ Q)]\!]u(\delta_1^+, \delta_2^+, a^+, c^-) = [\![b \leftarrow P;\ Q]\!]u(\delta_1^+, \delta_2^+, a^+, c^-)$$

it is sufficient to show that $\pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}L = \pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}R$.

To show $\pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}L \subseteq \pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}R$, let $(\delta_1^-, \delta_2^-, a^-, (b_a^-, b_b^-), b^+, c^+) \in L$ be arbitrary. The case $b^+ = \bot$ is impossible: the b^+ component of $l(\delta_1^+, \delta_2^+, a^+, \bot, (b_a^-, b_b^-), c^-)$ is not $\bot$, but the definition of L requires it to be $\sqsubseteq \bot$. So $b^+ = (b_a^+, b_b^+)_\bot$ for some b_a^+ and b_b^+. By calculating $r(\delta_1^+, \delta_2^+, a^+, b_b^+, b_b^-, c^-)$ and using the definitions of L and R, we deduce that $(\delta_1^-, \delta_2^-, a^-, b_b^-, b_b^+, c^+) \in R$.

To show $\pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}R \subseteq \pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}L$, let $(\delta_1^-, \delta_2^-, a^-, b^-, b^+, c^+) \in R$ be arbitrary. A similar argument gives $(\delta_1^-, \delta_2^-, a^-, (a^-, b^-), (a^+, b^+)_\bot, c^+) \in L$, so $\pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}L = \pi_{\Delta_1^-,\Delta_2^-,a^-,c^+}R$. □

Dually, a provider can receive a channel:

$$\frac{\Psi\ ;\ \Delta, a : A \vdash P :: b : B}{\Psi\ ;\ \Delta \vdash a \leftarrow \mathsf{recv}\ b;\ P :: b : A \multimap B}\ (\multimap R) \qquad \frac{\Psi\ ;\ \Delta, b : B \vdash P :: c : C}{\Psi\ ;\ \Delta, a : A, b : A \multimap B \vdash \mathsf{send}\ b\ a;\ P :: c : C}\ (\multimap L)$$

$$\frac{\Xi \vdash A\ \mathsf{type}_\mathsf{s}^+ \quad \Xi \vdash B\ \mathsf{type}_\mathsf{s}^-}{\Xi \vdash A \multimap B\ \mathsf{ctype}_\mathsf{s}^-}\ (\mathrm{C}\multimap)$$

The different polarities of the premisses of (C⊸) causes the polarized aspects to differ slightly from eqs. (25) and (26). The clauses for (⊸L) and (⊸R) are analogous to those for ($\otimes L$) and ($\otimes R$).

$$[\![\Xi \vdash A \multimap B\ \mathsf{ctype}_\mathsf{s}]\!] = \left([\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!] \times [\![\Xi \vdash B\ \mathsf{type}_\mathsf{s}]\!]\right)_\bot, \tag{29}$$

$$[\![\Xi \vdash A \multimap B\ \mathsf{ctype}_\mathsf{s}]\!]^- = \left([\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!]^+ \times [\![\Xi \vdash B\ \mathsf{type}_\mathsf{s}]\!]^-\right)_\bot, \tag{30}$$

$$[\![\Xi \vdash A \multimap B\ \mathsf{ctype}_\mathsf{s}]\!]^+ = [\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!]^- \times [\![\Xi \vdash B\ \mathsf{type}_\mathsf{s}]\!]^+. \tag{31}$$

3.6 Value transmission

Processes can send and receive functional values over channels. Given a functional term M, the provider $\_ \leftarrow \mathsf{output}\ a\ M; P$ evaluates M to a value v, sends v over the channel a, and continues as P. If M diverges, then $\_ \leftarrow \mathsf{output}\ a\ M; P$ gets stuck and sends no message. The client $x \leftarrow \mathsf{input}\ a; Q$ receives this value over a and binds it to x in the continuation Q.

$$\frac{\Psi \Vdash M : \tau \quad \Psi\ ;\ \Delta \vdash P :: a : A}{\Psi\ ;\ \Delta \vdash \_ \leftarrow \mathsf{output}\ a\ M; P :: a : \tau \wedge A}\ (\wedge R) \qquad \frac{\Psi, x : \tau\ ;\ \Delta, a : A \vdash Q :: c : C}{\Psi\ ;\ \Delta, a : \tau \wedge A \vdash x \leftarrow \mathsf{input}\ a; Q :: c : C}\ (\wedge L)$$

$$\frac{\tau\ \mathsf{type}_\mathsf{f} \quad \Xi \vdash A\ \mathsf{type}_\mathsf{s}^+}{\Xi \vdash \tau \wedge A\ \mathsf{ctype}_\mathsf{s}^+}\ (\mathrm{C}\wedge)$$

The judgment $\tau\ \mathsf{type}_\mathsf{f}$ means τ is a well-formed functional type. It is defined in section 3.9. A communication of type $\tau \wedge A$ is a value $v \in [\![\tau]\!]$ and paired with a communication $a \in [\![A]\!]$. We use lifting to account for the potential lack of a communication. The data relating to the value travels on the positive portion of the channel, so it only appears in the positive aspect.

$$[\![\Xi \vdash \tau \wedge A\ \mathsf{ctype}_\mathsf{s}]\!] = \left([\![\tau]\!] \times [\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!]\right)_\bot, \tag{32}$$

$$[\![\Xi \vdash \tau \wedge A\ \mathsf{ctype}_\mathsf{s}]\!]^- = [\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!]^-, \tag{33}$$

$$[\![\Xi \vdash \tau \wedge A\ \mathsf{ctype}_\mathsf{s}]\!]^+ = \left([\![\tau]\!] \times [\![\Xi \vdash A\ \mathsf{type}_\mathsf{s}]\!]^+\right)_\bot, \tag{34}$$

The relating natural transformations for proposition 2.1 are determined by $\pi_{\tau\wedge A}^- = \mathsf{down} * \pi_2 * \pi_A^-$ and $\pi_{\tau\wedge A}^+ = \left(\mathrm{id}_{[\![\tau]\!]} \times \pi_A^+\right)_\bot$.

To send the term M on a, we evaluate it under the current environment u to get an element of $[\![\Psi \Vdash M : \tau]\!]u$. Divergence is represented by $\bot_{[\![\tau]\!]}$; the other elements represent values of type τ. If v

represents a value, then we pair it with the output of the continuation process P on a^+. Otherwise, the process transmits nothing. This gives the clause:

$$\begin{aligned}
&[\![\Psi \;;\; \Delta \vdash _ \leftarrow \mathsf{output}\ a\ M; P :: a : \tau \wedge A]\!]u : \\
&\qquad : [\![\Delta]\!]^+ \times \left(a^- : [\![A]\!]^-\right) \to [\![\Delta]\!]^- \times \left(a^+ : \left([\![\tau]\!] \times [\![A]\!]^+\right)_\perp\right) \\
&[\![\Psi \;;\; \Delta \vdash _ \leftarrow \mathsf{output}\ a\ M; P :: a : \tau \wedge A]\!]u(\delta^+, a^-) \\
&= \begin{cases} \perp & \text{if } [\![\Psi \Vdash M : \tau]\!]u = \perp \\ (\delta^-, (v, a^+)_\perp) & \text{if } [\![\Psi \Vdash M : \tau]\!]u = v \neq \perp, \end{cases}
\end{aligned} \tag{35}$$

where $[\![\Psi \;;\; \Delta \vdash P :: a : A]\!]u(\delta^+, a^-) = (\delta^-, a^+)$.

The client $x \leftarrow \mathsf{input}\ a; Q$ waits on the channel a, i.e., it is strict in the a^+ component. If a communication (v, α^+) arrives on a^+, the client binds v to x in the environment and continues as Q with the observation α^+ on a^+.

$$\begin{aligned}
&[\![\Psi \;;\; \Delta, a : \tau \wedge A \vdash x \leftarrow \mathsf{input}\ a; Q :: c : C]\!]u : \\
&\qquad : [\![\Delta]\!]^+ \times \left(a^+ : \left([\![\tau]\!] \times [\![A]\!]^+\right)_\perp\right) \times [\![c : C]\!]^- \to [\![\Delta]\!]^- \times \left(a^- : [\![A]\!]^-\right) \times [\![c : C]\!]^+ \\
&[\![\Psi \;;\; \Delta, a : \tau \wedge A \vdash x \leftarrow \mathsf{input}\ a; Q :: c : C]\!]u \\
&= \mathrm{strict}_{a^+}\left(\lambda\left(\delta^+, a^+ : \left(v, \alpha^+\right)_\perp, c^-\right).[\![\Psi, x : \tau \;;\; \Delta, a : A \vdash Q :: c : C]\!][u \mid x \mapsto v](\delta^+, \alpha^+, d^-)\right)
\end{aligned} \tag{36}$$

The environment $[u \mid x \mapsto v] \in [\![\Psi, x : \tau]\!]$ maps x to $v \in [\![\tau]\!]$ and y to $u(y)$ for all $y \in \Psi$.

The η-property for value transmission is subtle because the functional term M might diverge. The naïve equivalence $a \leftarrow P;\ [M/x]Q \equiv a \leftarrow (\_ \leftarrow \mathsf{output}\ a\ M; P);\ (x \leftarrow \mathsf{input}\ a; Q)$ is not true in general. If x does not appear free in Q, the substitution on the left has no effect and $a \leftarrow P;\ Q$ runs as usual. If M diverges, then $\_ \leftarrow \mathsf{output}\ a\ M; P$ gets stuck and nothing is sent on a. Because $x \leftarrow \mathsf{input}\ a; Q$ waits on a, this means the process on the right is stuck and produces no output.

The two processes in the naïve equivalence have equal denotations whenever M converges. Process equivalence requires the processes to have equal denotations under all environments u. For the naïve equivalence to hold, M must then converge under every environment $u \in [\![\Psi]\!]$. This justifies the statement of proposition 3.7. Its proof relies on proposition 3.8.

Proposition 3.7. *Consider processes $\Psi \;;\; \Delta_1 \vdash P :: a : A$ and $\Psi, x : \tau \;;\; \Delta_2, a : A \vdash Q :: c : C$. If $\Psi \Vdash M : \tau$ and $[\![\Psi \Vdash M : \tau]\!]u \neq \perp$ for all $u \in [\![\Psi]\!]$, then*

$$x \leftarrow P;\ [M/x]Q \equiv c \leftarrow (\_ \leftarrow \mathsf{output}\ c\ M; P);\ (x \leftarrow \mathsf{input}\ c; Q).$$

Proposition 3.8 (Substitution of terms). *Let $\Psi = x_1 : \tau_1, \ldots, x_n : \tau_n$ and assume $\Psi \Vdash N : \tau$ and $\Psi \;;\; \Delta \vdash P :: c : C$. Then for all choices of n terms $\Phi \Vdash M_i : \tau_i$ $(1 \le i \le n)$,*

$$[\![\Phi \Vdash [\vec{M}/\vec{x}]N : \tau]\!] = [\![\Psi \Vdash N : \tau]\!] \circ \langle [\![\Phi \Vdash M_i : \tau_i]\!] \mid 1 \le i \le n \rangle, \tag{37}$$

$$[\![\Phi \;;\; \Delta \vdash [\vec{M}/\vec{x}]P :: c : C]\!] = [\![\Psi \;;\; \Delta \vdash P :: c : C]\!] \circ \langle [\![\Phi \Vdash M_i : \tau_i]\!] \mid 1 \le i \le n \rangle. \tag{38}$$

Dually, a provider can receive functional values. The semantic development is symmetric.

$$\frac{\Psi, x : \tau \;;\; \Delta \vdash Q :: a : A}{\Psi \;;\; \Delta \vdash x \leftarrow \mathsf{input}\ a; Q :: a : \tau \supset A}\ (\supset R) \qquad \frac{\Psi \Vdash M : \tau \quad \Psi \;;\; \Delta, a : A \vdash P :: c : C}{\Psi \;;\; \Delta, a : \tau \supset A \vdash \_ \leftarrow \mathsf{output}\ a\ M; P :: c : C}\ (\supset L)$$

$$\frac{\tau\ \mathsf{type}_\mathrm{f} \quad \Xi \vdash A\ \mathsf{type}_\mathrm{s}^-}{\Xi \vdash \tau \supset A\ \mathsf{ctype}_\mathrm{s}^-}\ (\mathrm{C}\supset)$$

3.7 Type equivalences

Our equirecursive semantics uses definitional equality to fold and unfold recursive session types. Unfolding is given by the rule (E-ρ) and is interpreted by eq. (51) below:

$$\frac{}{\Xi \vdash \rho\alpha.A \equiv [\rho\alpha.A/\alpha]A} \text{ (E-}\rho\text{)}$$

The following three rules make definitional equality is an equivalence relation:

$$\frac{}{\Xi \vdash A \equiv A} \text{ (E-Refl)} \qquad \frac{\Xi \vdash A' \equiv A}{\Xi \vdash A \equiv A'} \text{ (E-Sym)} \qquad \frac{\Xi \vdash A \equiv B \quad \Xi \vdash B \equiv C}{\Xi \vdash A \equiv C} \text{ (E-Trans)}$$

The equivalence $\Xi \vdash A \equiv B$ denotes a natural isomorphism $[\![\Xi \vdash A]\!] \to [\![\Xi \vdash B]\!]$. The above three rules are respectively interpreted by the natural isomorphisms

$$[\![\Xi \vdash A \equiv A]\!] = \mathsf{id}_{[\![\Xi \vdash A \text{ type}_s]\!]}, \tag{39}$$

$$[\![\Xi \vdash A \equiv A']\!] = [\![\Xi \vdash A' \equiv A]\!]^{-1}, \tag{40}$$

$$[\![\Xi \vdash A \equiv C]\!] = [\![\Xi \vdash B \equiv C]\!] \circ [\![\Xi \vdash A \equiv B]\!]. \tag{41}$$

We can make $\equiv$ a congruence relation. To illustrate, consider the rule

$$\frac{\Xi \vdash A \equiv A' \quad \Xi \vdash B \equiv B'}{\Xi \vdash A \otimes B \equiv A' \otimes B'} \text{ (E-}\otimes\text{)}$$

We use the fact that functors preserve isomorphisms to define the natural isomorphism

$$[\![\Xi \vdash A \otimes B \equiv A' \otimes B']\!] = \left([\![\Xi \vdash A \equiv A']\!] \times [\![\Xi \vdash B \equiv B']\!]\right)_\perp. \tag{42}$$

The polarized versions of proposition 2.2 are

$$[\![\Xi \vdash A \otimes B \equiv A' \otimes B']\!]^- = [\![\Xi \vdash A \equiv A']\!]^- \times [\![\Xi \vdash B \equiv B']\!]^-, \tag{43}$$

$$[\![\Xi \vdash A \otimes B \equiv A' \otimes B']\!]^+ = \left([\![\Xi \vdash A \equiv A']\!]^+ \times [\![\Xi \vdash B \equiv B']\!]^+\right)_\perp. \tag{44}$$

We use an equivalence $\cdot \vdash A' \equiv A$ between closed types A and A' as follows:

$$\frac{\Psi\ ;\ \Delta, a : A' \vdash P :: b : B \quad \cdot \vdash A' \equiv A}{\Psi\ ;\ \Delta, a : A \vdash P :: b : B} \text{ (Equiv-L)} \qquad \frac{\Psi\ ;\ \Delta \vdash P :: a : A' \quad \cdot \vdash A' \equiv A}{\Psi\ ;\ \Delta \vdash P :: a : A} \text{ (Equiv-R)}$$

By proposition 2.2, there exist natural isomorphisms $[\![A' \equiv A]\!]^p : [\![A']\!]^p \to [\![A]\!]^p$ for $p \in \{-, +\}$. The rules (Equiv-L) and (Equiv-R) are respectively interpreted as:

$$\begin{aligned}&[\![\Psi\ ;\ a : A, \Delta \vdash P :: b : B]\!]u \\ &= \left(\left(a^- : [\![A' \equiv A]\!]^-\right) \times \mathsf{id}\right) \circ [\![\Psi\ ;\ \Delta, a : A' \vdash P :: b : B]\!]u \circ \left(\left(a^+ : \left([\![A' \equiv A]\!]^+\right)^{-1}\right) \times \mathsf{id}\right),\end{aligned} \tag{45}$$

$$\begin{aligned}&[\![\Psi\ ;\ \Delta \vdash P :: a : A]\!]u \\ &= \left(\left(a^+ : [\![A' \equiv A]\!]^+\right) \times \mathsf{id}\right) \circ [\![\Psi\ ;\ \Delta \vdash P :: a : A']\!]u \circ \left(\left(a^- : \left([\![A' \equiv A]\!]^-\right)^{-1}\right) \times \mathsf{id}\right).\end{aligned} \tag{46}$$

3.8 Recursive types

Recursive types are formed by the rule (Cρ). We require that the type variable α and A have the same polarity so that $\rho\alpha.A$ and its unfolding $[\rho\alpha.A/\alpha]A$ have the same polarity. Every contractive session type is a session type. The rule (TC) makes this explicit.

$$\frac{}{\Xi, \alpha \text{ type}_s^p \vdash \alpha \text{ type}_s^p} \text{ (TVar)} \qquad \frac{\Xi, \alpha \text{ type}_s^p \vdash A \text{ ctype}_s^p}{\Xi \vdash \rho\alpha.A \text{ ctype}_s^p} \text{ (C}\rho\text{)} \qquad \frac{\Xi \vdash A \text{ ctype}_s^p}{\Xi \vdash A \text{ type}_s^p} \text{ (TC)}$$

As usual, the variable rule corresponds to projection. The rule (TC) carries no semantic information.

$$[\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}}^{p} \vdash \alpha\ \mathsf{type}_{\mathsf{s}}^{p}]\!] = \pi_{\alpha}^{\Xi,\alpha} : \mathbf{M}^{\Xi,\alpha} \to \mathbf{M}, \tag{47}$$

$$[\![\Xi \vdash A\ \mathsf{type}_{\mathsf{s}}^{p}]\!] = [\![\Xi \vdash A\ \mathsf{ctype}_{\mathsf{s}}^{p}]\!]. \tag{48}$$

The substitution property (proposition 3.10) holds only if $[\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}} \vdash \alpha\ \mathsf{type}_{\mathsf{s}}]\!]^{p} = \pi_{\alpha}^{\Xi,\alpha}$ and $\pi_{\alpha}^{p} = \mathsf{id}$ for $p \in \{-,+\}$. We interpret types into the subcategory $\mathbf{M}$ to make $\langle \pi_{\alpha}^{-}, \pi_{\alpha}^{+} \rangle$ a natural embedding. This is because its associated family of projections $\{\sqcap : \pi_{\alpha}^{\Xi,\alpha} D \times \pi_{\alpha}^{\Xi,\alpha} D \to \pi_{\alpha}^{\Xi,\alpha} D\}_{D}$ is natural only when all morphisms are meet-homomorphisms. For (TC), let $\pi^{p}_{A\ \mathsf{type}_{\mathsf{s}}} = \pi^{p}_{A\ \mathsf{ctype}_{\mathsf{s}}}$.

We interpret (Cρ) by the parametrized solution of a recursive domain equation. Every locally continuous functor $G : \mathbf{M} \to \mathbf{M}$ has a canonical fixed point $\mathsf{FIX}(G)$ in $\mathbf{M}$. Given a locally continuous functor $F : \mathbf{M}^{\Xi} \times \mathbf{M} \to \mathbf{M}$, the mapping $D \mapsto \mathsf{FIX}(F(D,-))$ extends to a locally continuous functor $F^{\dagger} : \mathbf{M}^{\Xi} \to \mathbf{M}$ given by [Abramsky and Jung 1995, Proposition 5.2.7]. For $p \in \{-,+\}$ we let

$$[\![\Xi \vdash \rho\alpha.A\ \mathsf{ctype}_{\mathsf{s}}]\!] = [\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}} \vdash A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{\dagger}, \tag{49}$$

$$[\![\Xi \vdash \rho\alpha.A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{p} = \left([\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}} \vdash A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{p}\right)^{\dagger}. \tag{50}$$

There exists a canonical isomorphism $\mathsf{fold} : G(\mathsf{FIX}(G)) \to \mathsf{FIX}(G)$. The canonical isomorphisms $\mathsf{fold}_{D} : F\left(D, F^{\dagger}D\right) \to F^{\dagger}D$ assemble into a natural isomorphism $\mathsf{Fold}^{F} : F \circ \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, F^{\dagger} \rangle \to F^{\dagger}$. By proposition 3.10,

$$[\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}} \vdash A\ \mathsf{ctype}_{\mathsf{s}}]\!] \circ \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, [\![\Xi, \alpha\ \mathsf{type}_{\mathsf{s}} \vdash A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{\dagger} \rangle = [\![\Xi \vdash [\rho\alpha.A/\alpha]A\ \mathsf{ctype}_{\mathsf{s}}]\!].$$

Let $\mathsf{Unfold}^{F} = (\mathsf{Fold}^{F})^{-1}$. We interpret (E-$\rho$) (section 3.7) by the natural isomorphism

$$\mathsf{Unfold}^{[\![\Xi\alpha \vdash A]\!]} : [\![\Xi \vdash \rho\alpha.A\ \mathsf{ctype}_{\mathsf{s}}]\!] \to [\![\Xi \vdash [\rho\alpha.A/\alpha]A\ \mathsf{ctype}_{\mathsf{s}}]\!]. \tag{51}$$

The same observations give its polarized interpretations satisfying proposition 2.2. They are

$$\mathsf{Unfold}^{[\![\Xi\alpha \vdash A]\!]^{p}} : [\![\Xi \vdash \rho\alpha.A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{p} \to [\![\Xi \vdash [\rho\alpha.A/\alpha]A\ \mathsf{ctype}_{\mathsf{s}}]\!]^{p}. \tag{52}$$

The dagger operation used in eqs. (49) and (50) is functorial. The relating natural transformations for proposition 2.1 are determined by $\pi^{p}_{\Xi \vdash \rho\alpha.A} = (\pi^{p}_{\Xi, \alpha \vdash A})^{\dagger}$ for $p \in \{-,+\}$. We sketch the proof that $\langle \pi^{-}_{\Xi \vdash \rho\alpha.A}, \pi^{+}_{\Xi \vdash \rho\alpha.A} \rangle$ is a natural embedding. The definition of Fold^{F} is natural in F. This means that diagram 53 commutes for all $F, G : \mathbf{M}^{\Xi} \times \mathbf{M} \to \mathbf{M}$ and natural transformations $\eta : F \to G$:

$$\begin{array}{ccc} F \circ \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, F^{\dagger} \rangle & \xrightarrow{\mathsf{Fold}^{F}} & F^{\dagger} \\ {\scriptstyle \eta * \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, \eta^{\dagger} \rangle} \downarrow & & \downarrow {\scriptstyle \eta^{\dagger}} \\ G \circ \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, G^{\dagger} \rangle & \xrightarrow{\mathsf{Fold}^{G}} & G^{\dagger}. \end{array} \tag{53}$$

We can recognize $\eta^{\dagger}$ as a mediating morphism out of an initial algebra. Let $F = [\![\Xi, \alpha \vdash A\ \mathsf{ctype}_{\mathsf{s}}]\!]$. Let $[\mathbf{M}^{\Xi} \xrightarrow{\text{l.c.}} \mathbf{M}]$ be the category of locally continuous functors $\mathbf{M}^{\Xi} \to \mathbf{M}$ and natural transformations between them, and let

$$H = F \circ \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, - \rangle : [\mathbf{M}^{\Xi} \xrightarrow{\text{l.c.}} \mathbf{M}] \to [\mathbf{M}^{\Xi} \xrightarrow{\text{l.c.}} \mathbf{M}].$$

An H-**algebra** is a pair (A, α) where A is a locally continuous functor $\mathbf{M}^{\Xi} \to \mathbf{M}$ and $\alpha : H(A) \to A$ is a natural transformation. An H-**algebra homomorphism** $(A, \alpha) \to (B, \beta)$ to is a natural transformation $\delta : A \to B$ such that $\delta \circ \alpha = \beta \circ H(\delta)$. The left vertical morphism of diagram 53 factors as $\eta \langle \mathsf{id}_{\mathbf{M}^{\Xi}}, G^{\dagger} \rangle \circ H(\eta^{\dagger})$. This implies $\eta^{\dagger}$ is an H-algebra homomorphism from $(F^{\dagger}, \mathsf{Fold}^{F})$ to

the H-algebra $(G^\dagger, \mathsf{Fold}^G \circ \eta\langle \mathrm{id}_{\mathbf{M}^\Xi}, G^\dagger\rangle)$. By taking G to be $[\![\Xi, \alpha \vdash A\ \mathrm{ctype_s}]\!]^p$, η to be $\pi^p_{\Xi,\alpha\vdash A}$, and letting $\phi^p = \mathsf{Fold}^G \circ \eta\langle \mathrm{id}_{\mathbf{M}^\Xi}, G^\dagger\rangle$, this gives an H-algebra $([\![\Xi \vdash \rho\alpha.A]\!]^p, \phi^p)$ for $p \in \{-,+\}$.

The forgetful functor from the category of H-algebras to $[\mathbf{M}^\Xi \xrightarrow{\text{l.c.}} \mathbf{M}]$ creates limits [Hughes 2001, Theorem 1.2.4]. The product $P = ([\![\Xi \vdash \rho\alpha.A]\!]^-, \phi^-) \times ([\![\Xi \vdash \rho\alpha.A]\!]^+, \phi^+)$ of H-algebras is $([\![\Xi \vdash \rho\alpha.A]\!]^- \times [\![\Xi \vdash \rho\alpha.A]\!]^+, (\!|\phi^-, \phi^+|\!))$, where $(\!|\phi^-, \phi^+|\!)$ is the unique natural transformation such that $\phi^q \circ H(\pi) = \pi \circ (\!|\phi^-, \phi^+|\!)$ for $q \in \{-,+\}$. In particular, $(\!|\phi^-, \phi^+|\!)$ is a natural embedding. The natural transformation $\langle \pi^-_{\rho\alpha.A}, \pi^+_{\rho\alpha.A}\rangle$ is an H-algebra homomorphism from $(F^\dagger, \mathsf{Fold}^F)$ to P. The initial H-algebra is $(F^\dagger, \mathsf{Fold}^F)$. Given any other H-algebra (A, α), the unique H-algebra homomorphism from $(F^\dagger, \mathsf{Fold}^F)$ to (A, α) is a natural embedding $F^\dagger \to A$ whenever α is a natural embedding. It follows that $\langle \pi^-_{\rho\alpha.A}, \pi^+_{\rho\alpha.A}\rangle$ is a natural embedding.

3.9 The functional layer

The functional layer is the simply-typed λ-calculus with a call-by-value semantics and a fixed-point operator. Its only base type is the type of quoted processes, formed by (T{}):

$$\frac{\cdot \vdash A_i\ \mathrm{type_s} \quad (0 \le i \le n)}{\{a_0 : A_0 \leftarrow a_1 : A_1, \ldots, a_n : A_n\}\ \mathrm{type_f}} \text{ (T\{\})}$$

We abbreviate ordered lists using an overline. The values of type $\{a : A \leftarrow \overline{a_i : A_i}\}$ are quoted processes $\Psi\ ;\ \overline{a_i : A_i} \vdash P :: a : A$. The interpretation of (T{}) is

$$[\![\{a : A \leftarrow \overline{a_i : A_i}\}\ \mathrm{type_f}]\!] = [\![\overline{a_i : A_i} \vdash a : A]\!]_\perp \tag{54}$$

where we abbreviate $\left[[\![\Delta]\!]^+ \times [\![a : A]\!]^- \to [\![\Delta]\!]^+ \times [\![a : A]\!]^-\right]$ as $[\![\Delta \vdash a : A]\!]$. The distinction between the "two" bottom elements in $[\![\overline{a_i : A_i} \vdash a : A]\!]_\perp$ is semantically meaningful. The genuine bottom element denotes the absence of a value of type $\{a : A \leftarrow \overline{a_i : A_i}\}$. The lifted bottom element $\lambda x.\perp$ from $[\![\overline{a_i : A_i} \vdash a : A]\!]$ corresponds to a stuck process that produces no output.

The introduction form quotes processes:

$$\frac{\Psi\ ;\ \overline{a_i : A_i} \vdash P :: c : C}{\Psi \Vdash c \leftarrow \{P\} \leftarrow \overline{a_i} : \{c : C \leftarrow \overline{a_i : A_i}\}} \text{ (I-\{\})}$$

It is given by post-composition with up:

$$\begin{aligned} &[\![\Psi \Vdash c \leftarrow \{P\} \leftarrow \overline{a_i} : \{c : C \leftarrow \overline{a_i : A_i}\}]\!] : [\![\Psi]\!] \to [\![\overline{a_i : A_i} \vdash c : C]\!]_\perp \\ &[\![\Psi \Vdash c \leftarrow \{P\} \leftarrow \overline{a_i} : \{c : C \leftarrow \overline{a_i : A_i}\}]\!] = \mathsf{up} \circ [\![\Psi\ ;\ \overline{a_i : A_i} \vdash P :: c : C]\!]. \end{aligned} \tag{55}$$

It is important that we use the non-strict unit up in eq. (55) to distinguish between stuck processes and the absence of a value of type $\{a : A \leftarrow \overline{a_i : A_i}\}$.

Unquoting a value of type $\{a : A \leftarrow \Delta\}$ spawns a process. It is given by the elimination rule

$$\frac{\Psi \Vdash M : \{c : C \leftarrow \overline{a_i : A_i}\} \quad \Psi\ ;\ c : C, \Delta \vdash Q :: d : D}{\Psi\ ;\ \overline{a_i : A_i}, \Delta \vdash c \leftarrow \{M\} \leftarrow \overline{a_i}; Q :: d : D} \text{ (E-\{\})}$$

with interpretation

$$\begin{aligned} &[\![\Psi\ ;\ \overline{a_i : A_i}, \Delta \vdash c \leftarrow \{M\} \leftarrow \overline{a_i}; Q :: d : D]\!]u : [\![\overline{a_i : A_i}, \Delta \vdash d : D]\!] \\ &[\![\Psi\ ;\ \overline{a_i : A_i}, \Delta \vdash c \leftarrow \{M\} \leftarrow \overline{a_i}; Q :: d : D]\!]u \\ &= \left(\left(\mathsf{down}\left([\![\Psi \Vdash M : \{c : A \leftarrow \overline{a_i : A_i}\}]\!]u\right)\right) \times [\![\Psi\ ;\ \Delta, c : A \vdash Q :: d : D]\!]u\right)^\ddagger_{c^- \times c^+}. \end{aligned} \tag{56}$$

This interpretation closely mimics the interpretation (3) for (Cut). Two cases are possible when unquoting M. If $[\![\Psi \Vdash M : \{c : A \leftarrow \overline{a_i : A_i}\}]\!]u$ is $\bot$, then $\mathsf{down}(\bot)$ is the constant function $\lambda x.\bot$. Semantically, this represents the processes that never produces output. Otherwise, M is $p_\bot$, where p is the denotation of some quoted process P.

The interpretations (55) and (56) satisfy the following η-property, which follows easily from the fact that (up, down) is a section-retraction pair.

Proposition 3.9. *Consider the processes* $\Psi\ ;\ \overline{a_i : A_i} \vdash P :: c : A$ *and* $\Psi\ ;\ \Delta, c : A \vdash Q :: d : D$. *We have the semantic equivalence* $c \leftarrow P;\ Q \equiv c \leftarrow \{c \leftarrow \{P\} \leftarrow \overline{a}_i\} \leftarrow \overline{a}_i; Q$.

To enforce a call-by-value semantics, we interpret the arrow type using the strict function space object of ω-$\mathbf{aBC}_{\bot!}$:

$$\frac{\tau\ \mathsf{type_f} \quad \sigma\ \mathsf{type_f}}{\tau \to \sigma\ \mathsf{type_f}}\ (\mathrm{T}{\to}) \qquad [\![\tau \to \sigma\ \mathsf{type_f}]\!] = [[\![\tau\ \mathsf{type_f}]\!] \xrightarrow{\bot!} [\![\sigma\ \mathsf{type_f}]\!]]. \tag{57}$$

The typing rules for the functional layer are standard:

$$\frac{}{\Psi, x : \tau \Vdash x : \tau}\ (\text{F-Var}) \qquad \frac{\Psi, x : \tau \Vdash M : \tau}{\Psi \Vdash \mathsf{fix}\ x.M : \tau}\ (\text{F-Fix})$$

$$\frac{\Psi, x : \tau \Vdash M : \sigma}{\Psi \Vdash \lambda x : \tau.M : \tau \to \sigma}\ (\text{F-Fun}) \qquad \frac{\Psi \Vdash M : \tau \to \sigma \quad \Psi \Vdash N : \tau}{\Psi \Vdash MN : \sigma}\ (\text{F-App})$$

The rule (F-Var) is interpreted as a projection:

$$[\![\Psi, x : \tau \Vdash x : \tau]\!] = \pi^{\Psi,x}_x : [\![\Psi, x : \tau]\!] \to [\![\tau]\!]. \tag{58}$$

The call-by-value semantics is as in [Stoy 1977]. Abstraction and application are given by:

$$[\![\Psi \Vdash \lambda x : \tau.M : \tau \to \sigma]\!]u = \mathsf{strict}_{[\![\tau]\!]}\left(\lambda v \in [\![\tau]\!].[\![\Psi, x : \tau \Vdash M : \sigma]\!][u \mid x \mapsto]\right), \tag{59}$$

$$[\![\Psi \Vdash MN : \sigma]\!]u = [\![\Psi \Vdash M : \tau \to \sigma]\!]u([\![\Psi \Vdash N : \tau]\!]u). \tag{60}$$

The fixed-point operator is interpreted using the dagger operation defined in section 4:

$$[\![\Psi \Vdash \mathsf{fix}\ x.M : \tau]\!] = [\![\Psi, x : \tau \Vdash M : \tau]\!]^\dagger. \tag{61}$$

Equivalent interpretations using the semantic fixed-point operator and directed suprema are:

$$[\![\Psi \Vdash \mathsf{fix}\ x.M : \tau]\!]u = \mathsf{fix}(\lambda v \in [\![\tau]\!].[\![\Psi, x : \tau \Vdash M : \tau]\!](u \mid x \mapsto v)), \tag{62}$$

$$[\![\Psi \Vdash \mathsf{fix}\ x.M : \tau]\!]u = [\![\Psi, x : \tau \Vdash M : \tau]\!](u \mid x \mapsto \bigsqcup_{n \in \mathbb{N}}{}^{\uparrow}\, m^n(\bot_{[\![\tau]\!]})) \tag{63}$$

where $m(v) = [\![\Psi, x : \tau \Vdash M : \tau]\!](u \mid x \mapsto v)$.

3.10 Structural properties

Our semantics respects the structural rules. It trivially respects the exchange rule because we interpret structural contexts as indexed products. Propositions 3.8 and 3.10 state that our semantics for types, type equivalence, terms, and processes respects the substitution property.

Proposition 3.10 (Substitution). *Let* $\Xi = \alpha_1\ \mathsf{type}_\mathsf{s}^{p_1}, \ldots, \alpha_n\ \mathsf{type}_\mathsf{s}^{p_n}$ *and* Θ *be contexts, and assume* $\Theta \vdash A_i\ \mathsf{type}_\mathsf{s}^{p_i}$ *or* $\Theta \vdash A_i\ \mathsf{ctype}_\mathsf{s}^{p_i}$ *for* $1 \le i \le n$. *Abbreviate* $\Theta \vdash A_i\ \mathsf{type}_\mathsf{s}^{p_i}$ *or* $\Theta \vdash A_i\ \mathsf{ctype}_\mathsf{s}^{p_i}$, *as the case may be, by* $\Theta \vdash A_i$. *Let* p *range over* $\{-, +\}$ *and* $(\!|\cdot|\!)$ *over* $[\![\cdot]\!]$, $[\![\cdot]\!]^-$, *and* $[\![\cdot]\!]^+$.

(1) *If* $\Xi \vdash B\ \mathsf{type}_\mathsf{s}^q$, *then*

$$(\!|\Theta \vdash [\vec{A}/\vec{\alpha}]B\ \mathsf{type}_\mathsf{s}^q|\!) = (\!|\Xi \vdash B\ \mathsf{type}_\mathsf{s}^q|\!) \circ \langle \alpha_i : (\!|\Theta \vdash A_i|\!) \mid 1 \le i \le n \rangle,$$

$$\pi^p_{\Theta \vdash [\vec{A}/\vec{\alpha}]B\ \mathsf{type}_\mathsf{s}^q} = \pi^p_{\Xi \vdash A\ \mathsf{type}_\mathsf{s}^q} * \left\langle \pi^p_{\Theta \vdash A_i} \;\middle|\; 1 \le i \le n \right\rangle.$$

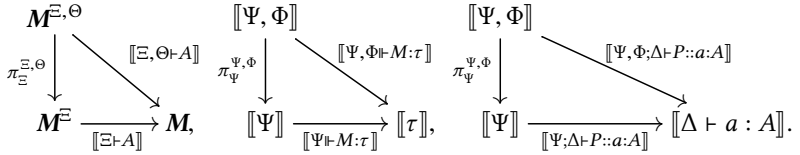
(2) *If* $\Xi \vdash B\ \mathrm{ctype}_s^q$, *then*

$$(\!|\Theta \vdash [\vec{A}/\vec{\alpha}]B\ \mathrm{ctype}_s^q|\!) = (\!|\Xi \vdash B\ \mathrm{ctype}_s^q|\!) \circ \langle \alpha_i : (\!|\Theta \vdash A_i|\!) \mid 1 \le i \le n\rangle,$$
$$\pi^p_{\Theta \vdash [\vec{A}/\vec{\alpha}]B\ \mathrm{ctype}_s^q} = \pi^p_{\Xi \vdash A\ \mathrm{ctype}_s^q} * \left\langle \pi^p_{\Theta \vdash A_i} \;\middle|\; 1 \le i \le n \right\rangle.$$

If $\Xi \vdash B \equiv C$, *then* $(\!|\Theta \vdash B \equiv C|\!) = (\!|\Xi \vdash B \equiv C|\!)\langle \alpha_i : (\!|\Theta \vdash A_i|\!) \mid 1 \le i \le n\rangle$.

It also respects weakening, a semantic property called coherence [Tennent 1995, p. 218].

Proposition 3.11 (Coherence). *If* Ξ, Θ *is a context of type variables and* $\Xi \vdash A$ *is a judgment* $\Xi \vdash A\ \mathrm{type}_s^p$ *or* $\Xi \vdash A\ \mathrm{ctype}_s^p$, *then the leftmost diagram commutes. It also commutes for the polarized interpretations* $[\![\cdot]\!]^-$ *and* $[\![\cdot]\!]^+$. *If* Ψ, Φ *is a context of functional variables and* $\Psi \Vdash M : \tau$ *(resp.,* $\Psi\ ;\ \Delta \vdash P :: a : A$*), then the centre (resp., rightmost) diagram commutes.*



(64)

Moreover, $\pi^p_{\Xi,\Phi \vdash A} = \pi^p_{\Xi \vdash A}\pi^{\Xi,\Phi}_{\Xi}$ *for* $p \in \{-,+\}$. *If* $\Xi \vdash A \equiv B$, *then* $[\![\Xi, \Phi \vdash A \equiv B]\!] = [\![\Xi \vdash A \equiv B]\!]\pi^{\Xi,\Phi}_{\Xi}$.

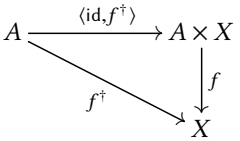
A consequence of the structural rules is that a recursive function is equivalent to its unfolding.

Corollary 3.12. *If* $\Psi, x : \tau \Vdash M : \tau$ *is a functional term, then* $[\mathsf{fix}\, x.M/x]M \equiv \mathsf{fix}\, x.M$.

Proof. Let $\Psi = \psi_1 : \tau_1, \dots, \psi_n : \tau_n$. By eq. (61), $[\![\Psi \Vdash \mathsf{fix}\, x.M : \tau]\!] = [\![\Psi, x : \tau \Vdash M : \tau]\!]^\dagger$. By the fixed-point identity (diagram 65), $[\![\Psi, x : \tau \Vdash M : \tau]\!]^\dagger = [\![\Psi, x : \tau \Vdash M : \tau]\!] \circ \langle \mathsf{id}_{[\![\Psi]\!]}, [\![\Psi \Vdash \mathsf{fix}\, x.M : \tau]\!]\rangle$. By eq. (58), this equals $[\![\Psi, x : \tau \Vdash M : \tau]\!] \circ \langle [\![\Psi \Vdash \psi_1 : \tau_1]\!], \dots, [\![\Psi \Vdash \psi_n : \tau_n]\!], [\![\Psi \Vdash \mathsf{fix}\, x.M : \tau]\!]\rangle$, which by proposition 3.8 is $[\![\Psi \Vdash [\mathsf{fix}\, x.M/x]M : \tau]\!]$. □

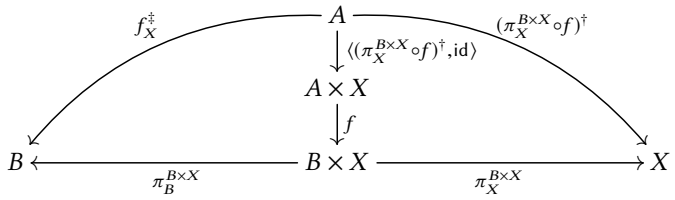
4 PROPERTIES OF DOUBLE-DAGGERS

The category ω-**aBC** has a continuous least-fixed-point operator $\mathsf{fix} : [D \to D] \to D$ for each object D. It also has a continuous fixed-point operator $(\cdot)^\dagger : [A \times X \to X] \to [A \to X]$ given by $f^\dagger(a) = \mathsf{fix}(\lambda x.f(a, x))$. It satisfies the fixed-point identity of [Bloom and Ésik 1996]:



(65)

We can also fix the X component of a morphism $f : A \times X \to B \times X$ to get a morphism $f^\ddagger_X : A \to B$. It is given by $f^\ddagger_X = \pi^{B\times X}_B \circ f \circ \langle (\pi^{B\times X}_X \circ f)^\dagger, \mathsf{id}\rangle$. If we think of f as a circuit with two inputs and two outputs, we can imagine connecting the X output to the X input. Inputting a on A and waiting for the X feedback loop to stabilize produces $f^\ddagger_X(a)$ on the B output wire. The function $f^\ddagger_X$ satisfies:



Though this definition is operationally intuitive, it can be intractable. The following proposition gives a more tractable formulation. By treating all products as indexed, we can implicitly reassociate and commute them as needed. This means, e.g., that we identify $A \times (B \times C)$ and $(C \times A) \times B$.

Proposition 4.1. *For all $f : A \times X \to B \times X$, we have $f \circ \langle (\pi_X^{B\times X} \circ f)^\dagger, \mathrm{id}\rangle = (f \circ \pi_{A\times X}^{B\times X\times A})^\dagger$. Consequently, $f_X^\ddagger = \pi_B^{B\times X} \circ (f \circ \pi_{A\times X}^{B\times X\times A})^\dagger$.*

We use proposition 4.1 to give two different approaches to compute $f_X^\ddagger$. Recall that given a domain A and a continuous $g : X \to X$, we can compute $\mathrm{fix}(g)$ in two ways:

- using the Kleene fixed-point theorem, with $\mathrm{fix}(g) = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} g^n(\bot_X)$;
- using a variant of the Knaster-Tarski theorem, with $\mathrm{fix}(g) = \bigsqcap\{x \in X \mid g(x) \sqsubseteq x\}$.

If $g : A \times X \to X$, then we can compute the $g^\dagger$ in two ways:

- using the Kleene fixed-point theorem, with $g^\dagger(a) = \bigsqcup_{n\in\mathbb{N}}^{\uparrow}(\lambda x \in X.g(a,x))^n(\bot_X)$;
- using a variant of the Knaster-Tarski theorem, with $g^\dagger(a) = \bigsqcap\{x \in X \mid g(a,x) \sqsubseteq x\}$.

This means the double dagger $f_X^\ddagger$ of a morphism $f : A \times X \to B \times X$ can be computed in two ways:

- using the Kleene fixed-point theorem, with $f_X^\ddagger(a) = \pi_B^{B\times X}\left(\bigsqcup_{n\in\mathbb{N}}^{\uparrow}(\lambda\,(b,x)\,.f\,(a,x))^n\,(\bot_B,\bot_X)\right)$;
- using a variant of the Knaster-Tarski theorem, with

$$f_X^\ddagger(a) = \pi_B^{B\times X}\left(\bigsqcap\{(b,x) \in B \times X \mid f(a,x) \sqsubseteq (b,x)\}\right). \tag{66}$$

Propositions 4.2 and 4.3 allow us to contract or expand the scope of the fixed-point operation.

Proposition 4.2. *For all $f : A\times X \to B\times X$ and $g : Y \to Z$, $(f \times g)_X^\ddagger = f_X^\ddagger \times g : A \times Y \to B \times Z$.*

Proposition 4.3. *Let $f : X \to X$ and $g : A \to B$. Then $(f \times g)_X^\ddagger = g$.*

Proposition 4.4 collapses nested fixed points into a single fixed point.

Proposition 4.4. *For all $f : A \times X \times Y \to B \times X \times Y$, $(f_X^\ddagger)_Y^\ddagger = f_{X\times Y}^\ddagger = (f_Y^\ddagger)_X^\ddagger : A \to B$.*

Proposition 4.5 is our analog of the parameter identity [Bloom and Ésik 1996, p. 8].

Proposition 4.5. *Let $s : X \to Y$ and $r : Y \to X$ with r strict and $r \circ s = \mathrm{id}_X$, and let $e : A \to B$, $f : B \times X \to C \times X$, and $g : C \to D$. Then $((g \times s) \circ f \circ (e \times r))_Y^\ddagger = g \circ f_X^\ddagger \circ e$.*

Dropping the strictness operator is often useful in computing semantic equivalences.

Proposition 4.6. *Let $f : A \times X \to Y \times C$ and $g : B \times Y \to D \times X$. The equality*

$$(\mathrm{strict}_X(f) \times g)_{X\times Y}^\ddagger = (f \times g)_{X\times Y}^\ddagger : A \times B \to C \times D$$

holds whenever $\pi_X^{D\times X} \circ g$ is not strict or $f = \mathrm{strict}_X(f)$.

5 CASE STUDY: BIT STREAMS

We illustrate our semantics by studying the recursive process F from the introduction. The tail call programming pattern is useful for defining recursive processes. To use this pattern, we introduce the syntactic sugar $\Psi\ ;\ \overline{a_i : A_i} \vdash d \leftarrow \{M\} \leftarrow \overline{a}_i :: d : A$ for the process

$$\frac{\Psi \Vdash M : \{c : A \leftarrow \overline{a_i : A_i}\} \quad \overline{\Psi\ ;\ c : A \vdash d \leftarrow c :: d : A}\ (\textsc{Fwd})}{\Psi\ ;\ \overline{a_i : A_i} \vdash c \leftarrow \{M\} \leftarrow \overline{a}_i; d \leftarrow c :: d : A}\ (\text{E-}\{\})$$

This gives the derived rule

$$\frac{\Psi \Vdash M : \{d : A \leftarrow \overline{a_i : A_i}\}}{\Psi\ ;\ \overline{a_i : A_i} \vdash d \leftarrow \{M\} \leftarrow \overline{a}_i :: d : A}\ (\textsc{Tail})$$

Using eqs. (2) and (56), we calculate that its denotation is

$$[\![\Psi \mathrel{;} \overline{a_i : A_i} \vdash d \leftarrow \{M\} \leftarrow \overline{a}_i :: d : A]\!] = \mathsf{down} \circ [\![\Psi \Vdash M : \{d : A \leftarrow \overline{a_i : A_i}\}]\!]. \tag{67}$$

For concreteness, we show that flipping a stream of `0` bits is equivalent to providing a stream of `1` bits. We encode bit streams using the session type $\mathtt{bits} = \rho\beta. \oplus\{\mathtt{0} : \beta, \mathtt{1} : \beta\}$. The stream of `0` bits is generated by the recursive quoted process Z.

```
· |- Z :: b : bits
b <- Z <- · = b.0; b <- Z <- ·
```

The quoted process uses no channels. It sends the label `0` and then uses a tail call to provide the tail of the bit stream. Let U be the symmetric recursive quoted process that provides the stream of `1` bits. Explicitly, we show that flipping the stream of `0` bits, $\cdot \mathrel{;} \cdot \vdash a \leftarrow \{Z\} \leftarrow \cdot; b \leftarrow \{F\} \leftarrow a :: b : \mathtt{bits}$, is semantically equivalent to $\cdot \mathrel{;} \cdot \vdash b \leftarrow \{U\} \leftarrow \cdot :: b : \mathtt{bits}$.

We begin by computing the denotation of Z. Denotations are defined by recursion on typing derivation, so we must first translate Z from concrete syntax to a well-typed term. Let $\mathtt{BITS} = \oplus\{\mathtt{0} : \mathtt{bits}, \mathtt{1} : \mathtt{bits}\}$ be the unfolding of the type `bits`, and let $\zeta = \{b : \mathtt{bits} \leftarrow \cdot\}$. Then Z is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Z : \zeta \Vdash Z : \zeta}\,(\text{F-Var})}{Z : \zeta \mathrel{;} \cdot \vdash b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}}\,(\text{Tail})}{Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{BITS}}\,(\oplus R_{\mathtt{0}}) \qquad \cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}}{Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}}\,(\text{Equiv-R})}{Z : \zeta \Vdash b \leftarrow \{b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot\} \leftarrow \cdot : \zeta}\,(\text{I-}\{\})}{\cdot \Vdash \mathsf{fix}\, Z.b \leftarrow \{b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot\} \leftarrow \cdot : \zeta}\,(\text{F-Fix})$$

We calculate the denotation step-by-step. By eqs. (58) and (67), we have

$$[\![Z : \zeta \mathrel{;} \cdot \vdash b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}]\!](Z : z) = \mathsf{down} \circ z.$$

By the interpretation (19) of $(\oplus R_{\mathtt{0}})$,

$$[\![Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{BITS}]\!](Z : z)\left(b^- : \left(b_{\mathtt{0}}^-, b_{\mathtt{1}}^-\right)\right) = \left(b^+ : \left(\mathtt{0}, \mathsf{down}\left(z\left(b_{\mathtt{0}}^-\right)\right)\right)_\perp\right).$$

The interpretation (46) of (Equiv-R) then gives

$$\begin{aligned}&[\![Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}]\!](Z : z)\\ &= \left(b^+ : [\![\cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}]\!]^+\right) \circ\\ &\qquad \circ [\![Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{BITS}]\!](Z : z) \circ \left(b^- : [\![\cdot \vdash \mathtt{bits} \equiv \mathtt{BITS}]\!]^-\right).\end{aligned}$$

The polarized interpretations of $\cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}$ are the canonical natural isomorphisms

$$\begin{aligned}[\![\cdot \vdash \mathtt{bits} \equiv \mathtt{BITS}]\!]^- &= \mathsf{Unfold}^- : [\![\mathtt{bits}]\!]^- \to \left(\left(\mathtt{0} : [\![\mathtt{bits}]\!]^-\right) \times \left(\mathtt{1} : [\![\mathtt{bits}]\!]^-\right)\right),\\ [\![\cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}]\!]^+ &= \mathsf{Fold}^+ : \left(\left(\mathtt{0} : [\![\mathtt{bits}]\!]^+_\perp\right) \oplus \left(\mathtt{1} : [\![\mathtt{bits}]\!]^+_\perp\right)\right) \to [\![\mathtt{bits}]\!]^+.\end{aligned}$$

Denote the semantic folding operation by $a \mapsto \ulcorner a \urcorner$ and unfolding by $\ulcorner a \urcorner \mapsto a$, then

$$[\![Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}]\!](Z : z)\left(b^- : \ulcorner\left(b_{\mathtt{0}}^-, b_{\mathtt{1}}^-\right)\urcorner\right) = \left(b^+ : \ulcorner\left(\mathtt{0}, \mathsf{down}\left(z\left(b_{\mathtt{0}}^-\right)\right)\right)_\perp\urcorner\right).$$

Finally, the interpretations (55) and (62) of (I-{}) and (F-Fix) give

$$\begin{aligned}&[\![\cdot \Vdash \mathsf{fix}\, Z.b \leftarrow \{b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot\} \leftarrow \cdot : \zeta]\!](\cdot)\\ &= \mathsf{fix}\left(\lambda z \in [\![\zeta]\!].\mathsf{up}\left([\![Z : \zeta \mathrel{;} \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}]\!](Z : z)\right)\right).\end{aligned}$$

Given continuous functions $f : B \to B$, $s : B \to A$, and $r : A \to B$, we have $\mathsf{fix}(s \circ f \circ r) = s(\mathsf{fix}(f))$ whenever r is strict and $r \circ s = \mathsf{id}$. Observe that down is strict, $\mathsf{down} \circ \mathsf{up} = \mathsf{id}$, and

$$\begin{aligned}
&\lambda z \in [\![\zeta]\!].\mathsf{up}\left([\![Z : \zeta\ ;\ \cdot \vdash b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot :: b : \mathtt{bits}]\!](Z : z)\right)\\
&= \lambda z \in [\![\zeta]\!].\mathsf{up}\left(\lambda\left(b^- : \ulcorner\left(b_{\mathtt{0}}^-, b_{\mathtt{1}}^-\right)\urcorner\right) \in [\![\mathtt{bits}]\!]^-.\left(b^+ : \ulcorner\left(\mathtt{0}, \mathsf{down}\left(z\left(b_{\mathtt{0}}^-\right)\right)\right)_\perp\urcorner\right)\right)\\
&= \mathsf{up} \circ \left(\lambda z \in [\![\cdot \vdash b : \mathtt{bits}]\!].\lambda\left(b^- : \ulcorner\left(b_{\mathtt{0}}^-, b_{\mathtt{1}}^-\right)\urcorner\right) \in [\![\mathtt{bits}]\!]^-.\left(b^+ : \ulcorner\left(\mathtt{0}, \left(z\left(b_{\mathtt{0}}^-\right)\right)_\perp\urcorner\right)\right) \circ \mathsf{down}.\right.
\end{aligned}$$

These facts imply

$$\begin{aligned}
&[\![\cdot \Vdash \mathsf{fix}\, Z.b \leftarrow \{b.\mathtt{0}; b \leftarrow \{Z\} \leftarrow \cdot\} \leftarrow \cdot : \zeta]\!](\cdot)\\
&= \mathsf{up}\left(\mathsf{fix}\left(\lambda z \in [\![\cdot \vdash b : \mathtt{bits}]\!].\lambda\left(b^- : \ulcorner\left(b_{\mathtt{0}}^-, b_{\mathtt{1}}^-\right)\urcorner\right) \in [\![\mathtt{bits}]\!]^-.\left(b^+ : \ulcorner\left(\mathtt{0}, \left(z\left(b_{\mathtt{0}}^-\right)\right)_\perp\urcorner\right)\right)\right)\right)\\
&= \mathsf{up}\left(\lambda b^- \in [\![\mathtt{bits}]\!]^-.\left(b^+ : \ulcorner\left(\mathtt{0}, \ulcorner\left(\mathtt{0}, \ulcorner\left(\mathtt{0}, \ldots\right)_\perp\urcorner\right)_\perp\urcorner\right)_\perp\urcorner\right)\right).
\end{aligned}$$

This confirms our intuition that Z denotes a quoted process that provides the stream $\mathbf{Z}$ of $\mathtt{0}$ bits. Let $\mathbf{U} = \ulcorner\left(\mathtt{1}, \ulcorner\left(\mathtt{1}, \ldots\right)_\perp\urcorner\right)_\perp\urcorner$ be the stream of 1 bits. By a symmetric argument, U denotes $(\lambda b^-.\mathbf{U})_\perp$.

We now compute the denotation of the quoted recursive process F from the introduction. Let $\phi = \{a : \mathtt{bits} \leftarrow b : \mathtt{bits}\}$. The branch $B_{\mathtt{0}}$ of F's case statement is given by

$$\dfrac{\dfrac{\dfrac{}{F : \phi \Vdash F : \phi}\ (\textsc{F-Var})}{F : \phi\ ;\ b : \mathtt{bits} \vdash a \leftarrow \{F\} \leftarrow b :: a : \mathtt{bits}}\ (\textsc{Tail})}{F : \phi\ ;\ b : \mathtt{bits} \vdash a.\mathtt{1}; a \leftarrow \{F\} \leftarrow b :: a : \mathtt{BITS}}\ (\oplus R_{\mathtt{1}})$$

The branch $B_{\mathtt{1}}$ is symmetric. The functional term F is then given by

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{F : \phi\ ;\ b : \mathtt{bits} \vdash B_l :: a : \mathtt{BITS}\quad (\forall l \in \{\mathtt{0},\mathtt{1}\})}{F : \phi\ ;\ b : \mathtt{BITS} \vdash \mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}} :: a : \mathtt{BITS}}\ (\oplus\mathrm{L}) \quad \cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}}{F : \phi\ ;\ b : \mathtt{bits} \vdash \mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}} :: a : \mathtt{BITS}}\ (\textsc{Equiv-L}) \quad \cdot \vdash \mathtt{BITS} \equiv \mathtt{bits}}{F : \phi\ ;\ b : \mathtt{bits} \vdash \mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}} :: a : \mathtt{bits}}\ (\textsc{Equiv-R})}{F : \phi \Vdash a \leftarrow \{\mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}}\} \leftarrow b : \phi}\ (\text{I-}\{\})}{\cdot \Vdash \mathsf{fix}\, F.a \leftarrow \{\mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}}\} \leftarrow b : \phi}\ (\text{T-Fix})$$

We calculate that the denotation of the branch $B_{\mathtt{0}}$ is

$$[\![F : \phi\ ;\ b : \mathtt{bits} \vdash a.\mathtt{1}; t \leftarrow \{F\} \leftarrow b :: a : \mathtt{BITS}]\!](F : f)\left(b^+, \left(a_{\mathtt{0}}^-, a_{\mathtt{1}}^-\right)\right) = \left(b^-, \left(\mathtt{1}, a^+\right)_\perp\right)$$

where $(b^-, a^+) = \mathsf{down}\,(f)\,(b^+, a_{\mathtt{1}}^-)$. The denotation of the branch $B_{\mathtt{1}}$ is symmetric. Then

$$\begin{aligned}
&[\![F : \phi\ ;\ b : \mathtt{bits} \vdash \mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}} :: a : \mathtt{bits}]\!](F : f)\\
&= \mathrm{strict}_{b^+}\left(\lambda(\ulcorner b^+\urcorner, \ulcorner\left(a_{\mathtt{0}}^-, a_{\mathtt{1}}^-\right)\urcorner).\begin{cases}\left(b^- : \ulcorner\left(\mathtt{0} : b_{\mathtt{0}}^-, \mathtt{1} : \perp\right)\urcorner, f^+ : \ulcorner\left(\mathtt{1}, a_{\mathtt{0}}^+\right)_\perp\urcorner\right) & \text{if } b^+ = \left(\mathtt{0}, b_{\mathtt{0}}^+\right)_\perp\\ \left(b^- : \ulcorner\left(\mathtt{1} : \perp, \mathtt{1} : b_{\mathtt{1}}^-\right)\urcorner, f^+ : \ulcorner\left(\mathtt{0}, a_{\mathtt{1}}^+\right)_\perp\urcorner\right) & \text{if } b^+ = \left(\mathtt{1}, b_{\mathtt{1}}^+\right)_\perp\end{cases}\right)
\end{aligned}$$

where $(b_l^-, a_l^+) = \mathsf{down}\,(f)\,(b_l^+, a_l^-)$ for $l \in L$. The quoted process denotes

$$\begin{aligned}
&[\![F : \phi \Vdash a \leftarrow \{\mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}}\} \leftarrow b : \phi]\!]\\
&= \mathsf{up} \circ [\![F : \phi\ ;\ b : \mathtt{bits} \vdash \mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}} :: a : \mathtt{bits}]\!]\\
&= \mathsf{up} \circ \Phi \circ \mathsf{down}
\end{aligned}$$

where $\Phi : [\![b : \mathtt{bits} \vdash a : \mathtt{bits}]\!] \to [\![b : \mathtt{bits} \vdash a : \mathtt{bits}]\!]$ is

$$\Phi(r) = \mathrm{strict}_{b^+}\left(\lambda(\ulcorner b^+\urcorner, \ulcorner\left(a_{\mathtt{0}}^-, a_{\mathtt{1}}^-\right)\urcorner).\begin{cases}\left(\ulcorner\left(\mathtt{0} : b_{\mathtt{0}}^-, \mathtt{1} : \perp\right)\urcorner, \ulcorner\left(\mathtt{1}, a_{\mathtt{0}}^+\right)_\perp\urcorner\right) & \text{if } b^+ = \left(\mathtt{0}, b_{\mathtt{0}}^+\right)_\perp\\ \left(\ulcorner\left(\mathtt{0} : \perp, \mathtt{1} : b_{\mathtt{1}}^-\right)\urcorner, \ulcorner\left(\mathtt{0}, a_{\mathtt{1}}^+\right)_\perp\urcorner\right) & \text{if } b^+ = \left(\mathtt{1}, b_{\mathtt{1}}^+\right)_\perp\\ \text{where } r(b_l^+, a_l^-) = (b_l^-, a_l^+) \text{ for } l \in L.\end{cases}\right).$$

By the above remarks on section-retraction pairs, the denotation of the recursive quoted process is:

$$[\![\cdot \Vdash \mathsf{fix}\, F.a \leftarrow \{\mathsf{case}\ b\ \{l \Rightarrow B_l\}_{l\in\{\mathtt{0},\mathtt{1}\}}\} \leftarrow b : \phi]\!](\cdot) = \mathsf{up}(\mathsf{fix}(\Phi)).$$

Consider the composition $\cdot\,;\cdot \vdash b \leftarrow \{Z\} \leftarrow \cdot; a \leftarrow \{F\} \leftarrow b :: a : \mathtt{bits}$. By eqs. (56) and (67),

$$\begin{aligned}
&[\![\cdot\,;\cdot \vdash b \leftarrow \{Z\} \leftarrow \cdot; a \leftarrow \{F\} \leftarrow b :: a : \mathtt{bits}]\!](\cdot)(a^-)\\
&= \left(\mathsf{down}\left([\![\cdot \Vdash Z : \{b : \mathtt{bits} \leftarrow \cdot\}]\!](\cdot)\right) \times \mathsf{down}\left([\![\cdot \Vdash F : \{a : \mathtt{bits} \leftarrow b : \mathtt{bits}\}]\!](\cdot)\right)\right)^{\ddagger}_{b^-\times b^+}(a^-)\\
&= \left(\left(\lambda b^- \in [\![\mathtt{bits}]\!]^-.\, (b^+ : \mathbf{Z})\right)\right) \times \mathsf{fix}(\Phi)\Big)^{\ddagger}_{b^-\times b^+}(a^-)\\
&= \pi_{a^+}\left(\bigsqcap \left\{(a^+ : \alpha^+, b^- : \beta^-, b^+ : \beta^+) \mid \left(\left(\lambda b^-.(b^+ : \mathbf{Z})\right) \times \mathsf{fix}(\Phi)\right)(a^-, \beta^-, \beta^+) \sqsubseteq (\alpha^+, \beta^-, \beta^+)\right\}\right)
\end{aligned}$$

The value $\mathbf{Z}$ is maximal, so β^+ must always be $\mathbf{Z}$. The value b^- is unconstrained. Meets of products are computed component-wise, so the above is

$$= \pi_{a^+}\left(\bigsqcap \left\{(a^+ : \alpha^+, b^- : \beta^-) \mid \mathsf{fix}(\Phi)(a^-, \mathbf{Z}) \sqsubseteq (\alpha^+, \beta^-)\right\}\right) = \pi_{a^+}\left(\mathsf{fix}(\Phi)(a^-, \mathbf{Z})\right).$$

We use continuity to show that $\pi_{a^+}(\mathsf{fix}(\Phi)(a^-, \mathbf{Z})) = \mathbf{U}$ for all $a^- \in [\![\mathtt{bits}^-]\!]$. The finite prefixes $\mathbf{U}_n$ of $\mathbf{U}$ are inductively defined by $\mathbf{U}_0 = \bot$ and $\mathbf{U}_{n+1} = \ulcorner(1, \mathbf{U}_n)_\bot\urcorner$. A symmetric definition gives the finite prefixes the $\mathbf{Z}_n$ of $\mathbf{Z}$. They satisfy $\mathbf{U} = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \mathbf{U}_n$ and $\mathbf{Z} = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} \mathbf{Z}_n$. An induction on n shows that $\pi_{a^+}(\Phi^n(\bot)(\alpha^-, \mathbf{Z}_n)) = \mathbf{U}_n$ for all $n \in \mathbb{N}$ and $\alpha^- \in [\![\mathtt{bits}]\!]^-$. Continuity then implies

$$\pi_{a^+}(\mathsf{fix}(\Phi)(a^-, \mathbf{Z})) = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \pi_{a^+}(\Phi^n(\bot)(\alpha^-, \mathbf{Z}_n)) = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} \mathbf{U}_n = \mathbf{U}.$$

This implies for all $\alpha^- \in [\![\mathtt{bits}]\!]^-$ and $u \in [\![\cdot]\!] = \{(\cdot)\}$ that

$$[\![\cdot\,;\cdot \vdash b \leftarrow \{Z\} \leftarrow \cdot; a \leftarrow \{F\} \leftarrow b :: a : \mathtt{bits}]\!]u(a^-) = [\![\cdot\,;\cdot \vdash b \leftarrow \{U\} \leftarrow \cdot :: b : \mathtt{bits}]\!]u(a^-),$$

i.e., $a \leftarrow \{Z\} \leftarrow \cdot; b \leftarrow \{F\} \leftarrow a \equiv b \leftarrow \{U\} \leftarrow \cdot$.

6 RELATED AND FUTURE WORK

Honda [1993] and Takeuchi et al. [1994] introduced session types to describe sessions of interaction. Caires and Pfenning [2010] observed a proofs-as-programs correspondence between the session-typed π-calculus and intuitionistic linear logic, where the (Cut) rule captures process communication. Toninho et al. [2013] built on this correspondence and introduced the monadic integration between functional and message-passing programming. They specified their language's operational behaviour using a substructural operational semantics. Their functional layer supports polymorphic and inductive types. We conjecture that our semantics can be extended to support these features. Gay and Vasconcelos [2009] introduced asynchronous communication for session-typed languages. They used an operational semantics and buffers to model asynchronicity. Pfenning and Griffith [2015] observed that the polarity of a type determines the direction of communication along a channel. They observed that synchronous communication can be encoded in an asynchronous setting using explicit shift operators. They gave a computational interpretation to polarized adjoint logic. In this interpretation, linear propositions, affine propositions, and unrestricted propositions correspond to different modes in which resources can be used. As future work, we would like to extend our semantics to support this computational interpretation. We would also like to show that our denotational semantics is sound and adequate for an operational notion of observation based on the substructural operational semantics of Toninho et al. [2013] and Pfenning and Griffith [2015].

Wadler [2014] introduced "Classical Processes" (CP), a proofs-as-programs interpretation for classical linear logic that builds on the ideas of Caires and Pfenning [2010]. Atkey [2017] gave a denotational semantics for CP, where types are interpreted as sets and processes are interpreted as relations over these. Because processes in CP are proof terms for classical linear logic, the

interpretation of processes is identical to the relational semantics of proofs in classical linear logic [Barr 1991]. Our canonical interpretation of types builds heavily on these semantics. For example, we both interpret multiplicative connectives as tupling and additive connectives using disjoint unions. Our jump from sets and relations to domains and continuous functions was motivated by two factors. First, domains provide a natural setting for studying recursion. Second, we believe that monotonicity and continuity are essential properties for a semantics of processes with infinite data, and it is unclear how to capture these properties in a relational setting. Our transition to domains and functions required polarized interpretations of types. In the case of recursive types, defining the relating natural transformations for proposition 2.1 and showing that they satisfied the structural rules required significant generalizations of the techniques found in [Smyth and Plotkin 1982]. Atkey interpreted processes composition as relational composition. Our interpretation of composition is more complex, but we believe the identities in section 4 make it tractable.

Castellan and Yoshida [2019] gave a game semantics interpretation of the session π-calculus with recursion. It is fully abstract relative a barbed congruence notion of behavioural equivalence. They interpreted session types as event structures that encode games. These event structures are endowed with an ω-cpo structure. They then interpreted open types as continuous maps between these and recursive types as least fixed points. Open processes are interpreted as continuous maps that describe strategies. We conjecture that our semantics could be related via barbed congruence.

Kokke et al. [2019] introduced "hypersequent classical processes" (HCP). HCP is a revised proofs-as-processes interpretation between classical linear logic and the π-calculus. Building on Atkey's semantics for CP, they gave HCP a denotational semantics using Brzozowski derivatives [Brzozowski 1964]. Their denotational semantics is fully abstract relative to their notions of bisimilarity and barbed congruence. It does not handle recursion or the transmission of functional values.

Pérez et al. [2012; 2014] introduced a theory of logical relations for session-typed processes. They also introduced a bisimulation-based notion of observational equivalence and showed that proof conversions are sound with respect to observational equivalence. As future work, we would like to investigate the correspondence between our semantic equivalence and their notions of equivalence.

Gay and Hole [2005] gave a subtyping system for session types. We would like to investigate its implications for polarized SILL using our semantics. We conjecture that subtyping judgments $\Xi \vdash A \leq B$ will denote natural embeddings $[\![\Xi \vdash A]\!] \to [\![\Xi \vdash B]\!]$ satisfying a variant of proposition 2.2.

ACKNOWLEDGMENTS

This work is funded in part by a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship. The author thanks Stephen Brookes and Frank Pfenning for their comments.

Last updated: August 20, 2019.

REFERENCES

Samson Abramsky and Achim Jung. 1995. Domain Theory. In *Semantic Structures*, S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (Eds.). Vol. 3. Oxford University Press Inc., New York, 1–168.

Robert Atkey. 2017. Observed Communication Semantics for Classical Processes. In *Programming Languages and Systems (Lecture Notes in Computer Science)*, Hongseok Yang (Ed.). Springer Berlin Heidelberg, Berlin, 56–82. https://doi.org/10.1007/978-3-662-54434-1

Michael Barr. 1991. $*$-Autonomous Categories and Linear Logic. *Mathematical Structures in Computer Science* 1, 2 (1991), 159–178. https://doi.org/10.1017/s0960129500001274

Stephen L. Bloom and Zoltán Ésik. 1996. Fixed-Point Operations on ccc's. Part I. *Theoretical Computer Science* 155, 1 (1996), 1–38. https://doi.org/10.1016/0304-3975(95)00010-0

Janusz A. Brzozowski. 1964. Derivatives of Regular Expressions. *J. ACM* 11, 4 (1964), 481–494. https://doi.org/10.1145/321239.321249

Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 — Concurrency Theory (Lecture Notes in Computer Science)*, Paul Gastin and François Laroussinie (Eds.). Springer-Verlag Berlin Heidelberg, 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

Simon Castellan and Nobuko Yoshida. 2019. Two Sides of the Same Coin: Session Types and Game Semantics: A Synchronous Side and an Asynchronous Side. *Proc. ACM Program. Lang.* 3, POPL, Article 27 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290340

Roy L. Crole. 1993. *Categories for Types.* Cambridge University Press, Cambridge, United Kingdom.

Simon Gay and Malcolm Hole. 2005. Subtyping for Session Types in the Pi Calculus. *Acta Informatica* 42, 2-3 (11 2005), 191–225. https://doi.org/10.1007/s00236-005-0177-z

Simon J. Gay and Vasco T. Vasconcelos. 2009. Linear Type Theory for Asynchronous Session Types. *Journal of Functional Programming* 20, 1 (2009), 19–50. https://doi.org/10.1017/s0956796809990268

Carl A. Gunter. 1992. *Semantics of Programming Languages.* The MIT Press, Cambridge, Massachusetts.

Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR'93 (Lecture Notes in Computer Science)*, Eike Best (Ed.). Springer-Verlag Berlin Heidelberg, Berlin, 509–523. https://doi.org/10.1007/3-540-57208-2_35

Jesse Hughes. 2001. *A Study of Categories of Algebras and Coalgebras.* phdthesis. Carnegie Mellon University, Pittsburgh, Pennsylvania.

Wen Kokke, Fabrizio Montesi, and Marco Peressotti. 2019. Better Late Than Never: A Fully-abstract Semantics for Classical Processes. *Proc. ACM Program. Lang.* 3, POPL, Article 24 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290337

Frank Pfenning and Dennis Griffith. 2015. Polarized Substructural Session Types. In *Foundations of Software Science and Computation Structures (Lecture Notes in Computer Science)*, Andrew Pitts (Ed.). Springer-Verlag GmbH Berlin Heidelberg, Berlin Heidelberg, 3–32. https://doi.org/10.1007/978-3-662-46678-0_1

Benjamin Pierce. 2002. *Types and Programming Languages.* The MIT Press, Cambridge, Massachusetts.

Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. 2012. Linear Logical Relations for Session-Based Concurrency. In *Programming Languages and Systems (Lecture Notes in Computer Science)*, Helmut Seidl (Ed.). Springer-Verlag Berlin Heidelberg, Heidelberg, 539–558. https://doi.org/10.1007/978-3-642-28869-2_27

Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. 2014. Linear Logical Relations and Observational Equivalences for Session-Based Concurrency. *Information and Computation* 239 (2014), 254–302. https://doi.org/10.1016/j.ic.2014.08.001

John C. Reynolds. 2009. *Theories of Programming Languages.* Cambridge University Press, New York, New York.

Emily Riehl. 2016. *Category Theory in Context.* Dover Publications, Inc, Mineola, New York.

M. B. Smyth and G. D. Plotkin. 1982. The Category-Theoretic Solution of Recursive Domain Equations. *SIAM J. Comput.* 11, 4 (1982), 761–783. https://doi.org/10.1137/0211062

Joseph E. Stoy. 1977. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* The MIT Press, Cambridge, Massachusetts.

Kaku Takeuchi, Kohei Honda, and Makoto Kubo. 1994. An Interaction-Based Language and Its Typing System. In *PARLE'94 (Lecture Notes in Computer Science)*, Costas Halatsis, Dimitrios Maritsas, George Philokyprou, and Sergios Theodoridis (Eds.). Springer-Verlag Berlin Heidelberg, Berlin, 398–413. https://doi.org/10.1007/3-540-58184-7_118

R. D. Tennent. 1995. Denotational Semantics. In *Semantic Structures*, S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (Eds.). Vol. 3. Oxford University Press Inc., New York, 169–322.

Bernardo Toninho. 2015. *A Logical Foundation for Session-based Concurrent Computation.* Ph.D. Dissertation. Universidade Nova de Lisboa.

Bernardo Toninho, Luis Caires, and Frank Pfenning. 2013. Higher-Order Processes, Functions, and Sessions: A Monadic Integration. In *Programming Languages and Systems (Lecture Notes in Computer Science)*, Matthias Felleisen and Philippa Gardner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–369. https://doi.org/10.1007/978-3-642-37036-6_20

Philip Wadler. 2014. Propositions As Sessions. *Journal of Functional Programming* 24, 2-3 (2014), 384–418. https://doi.org/10.1017/s095679681400001x

Zoltán Ésik. 2009. Fixed Point Theory. In *Handbook of Weighted Automata*, Manfred Droste, Werner Kuich, and Heiko Vogler (Eds.). Springer-Verlag Berlin Heidelberg, 29–65. https://doi.org/10.1007/978-3-642-01492-5_2