Safe Session-Based Concurrency with Shared Linear State

Pedro Rocha and Luís Caires

We introduce CLASS, a session-typed, higher-order, core language that supports concurrent computation with shared linear state. We believe that CLASS is the first proposal for a foundational language able to flexibly express realistic concurrent programming idioms, with a type system ensuring all the following three key properties: CLASS programs never misuse or leak stateful resources or memory, they never deadlock, and they always terminate. CLASS owes these strong properties to a propositions-as-types foundation based on Linear Logic, which we conservatively extend with logically motivated constructs for shareable affine state. We illustrate CLASS expressiveness with several examples involving memory-efficient linked data structures, sharing of resources with linear usage protocols, and sophisticated thread synchronisation, which may be type-checked with a perhaps surprisingly light type annotation burden.

1 Introduction

Stateful programming involving concurrency and shared state plays a prominent role in modern software development, but, in practice, getting concurrent code right is still quite hard for common developers. Typical sources of "bugs" include resource leaks (forgetting to release unused memory or close a socket), violation of resource state preconditions (writing to a closed file or sending out-of-order messages), races (data invariant breaking, erratic sharing of resources), deadlocks (indefinite wait for lock release or incoming messages), livelocks, and even general non-termination. Fifty years ago Hoare noted [39]: "Parallel programs are particularly prone to time-dependent errors, which either cannot be detected by program testing nor by run-time checks. It is therefore very important that a high-level language designed for this purpose should provide complete security against time-dependent errors by means of a *compile-time* check". It does not come as a surprise that finding ways to approximate such certainly very ambitious goal is still today the object of exciting intense research.

In this paper, we approach this challenge by leveraging the propositions-as-types (PaT) paradigm towards the realm of concurrency and shared state. PaT is known to offer a unifying framework connecting logic, computation, and programming languages. Since the seminal work of Curry and Howard [40], it is a prolific structuring concept for designing and reasoning about programming languages (see [76]). Remarkably, languages derived within PaT intrinsically satisfy crucial properties: *type preservation* (since reduction corresponds to cut-reduction), *confluence* (since computation corresponds to proof simplification), *deadlock freedom* (as a consequence of cut-elimination) and *livelock freedom / termination* (as a consequence of strong normalisation).

Although PaT has a traditional focus on functional computation, the emergence of linear logic has progressively motivated interpretations of stateful/resourceful computation [72, 1, 14, 2, 12], eventually leading to the discovery of tight correspondences between session types and linear logic [21, 26, 75]. These systems already capture aspects of state change, namely in the sequential execution of session protocols, thus raising the question of whether such approaches could be extended to express notions of shared mutable state, subject to interference, as found in typical imperative and concurrent programs. Recently, such challenge was addressed by several works [9, 60, 62]. In particular, [62] developed a first basic shared state model enjoying all the aforementioned strong properties of PaT. However, although [62] supports higher-order shareable store for pure values of replicated type, it forbids linear objects, such as stateful processes or data structures with update in-place, to be stored and shared as in languages like Java, Rust, and in the CLASS core language we introduce herein.

In this work, we develop a novel, more fundamental approach to shared state and PaT, and introduce CLASS, a typed, higher-order, session based core language that supports general concurrent computation with dynamically allocated shared linear (more precisely, affine) state. We believe that CLASS is the first proposal for a foundational language. able to flexibly express realistic concurrent programming idioms, while ensuring all the following three key properties by static typing: CLASS programs never misuse or leak stateful resources or memory, they never deadlock, and they always terminate.

Despite the strength of its type system, CLASS expressiveness and effectiveness substantially overcomes limitations of related works, as we show with compelling program examples that can be algorithmically typed for memory safety, dead- and live-lock freedom with a perhaps surprisingly light type annotation burden. CLASS owes these strong properties to is PaT foundation based on Second-Order Linear Logic, already known to capture the polymorphic session calculus and the linear System F [68], but which we conservatively extend with novel logically motivated constructs for shareable affine state, also based on DiLL co-exponentials [34, 62], but to which we give here a different, more general and fundamental interpretation.

1.1 Overview

A main novelty and source of CLASS's expressiveness, flexibility and strong metatheoretical properties resides in its mechanism for shared state composition. It is interesting to overview such mechanism in the context of the basic composition and interaction principles of the fundamental linear logic interpretations [21, 26, 75]. Our computational model is structured around processes that interact via binary sessions, the basic composition rules being mix and cut.

$$\frac{P \vdash \Delta_1; \Gamma \quad Q \vdash \Delta_2; \Gamma}{P \parallel Q \vdash \Delta_1, \Delta_2; \Gamma}\,[\text{Tmix}] \qquad \frac{P \vdash \Delta_1, x : A; \Gamma \quad Q \vdash \Delta_2, x : \overline{A}; \Gamma}{P \mid x \mid Q \vdash \Delta_1, \Delta_2; \Gamma}\,[\text{Tcut}]$$

The mix rule types the independent composition of processes P and Q, which do not share any free names and run side-by-side without interacting. This is captured by the implicit disjointness of their linear typing contexts Δ_1 and Δ_2, declaring the types of their interaction channels. Interactive composition is expressed by the cut rule, which connects exactly two processes P and Q through a *single* linear session x with *dual typed* endpoints ($x : A$ and $x : \overline{A}$), following Abramsky's idea of "cut as interactive composition" [1].

Intuitively, duality of endpoint (session) types ensures that all interactions between P and Q on x always matches: when P sends, Q receives; when Q offers, P choses; and likewise for all types. Notice that sharing a single channel x between the threads P and Q is important to ensure acyclicity of proof structures, and cut-elimination/deadlock absence. But P, Q may use an arbitrary number of linear channels, in Δ_1, Δ_2, to also compose with other processes.

Shared composition in session types is available for *replicated* "server" objects $!x(y); P$, typed by the linear logic exponential type bang $!A$. Contraction of the dual exponential type why-not $?\overline{A}$ allows an unbounded number of usages of such replicated server object to be introduced in client processes. In the dyadic presentation of linear logic (cf. [5, 11]), contraction is expressed by moving ?-typed names into the unrestricted context Γ, with the [T?] rule.

$$\dfrac{!x(y); P \vdash x :!A; \Gamma \qquad \dfrac{Q \vdash \Delta; \Gamma, x : \overline{A}}{?x; Q \vdash \Delta, x :?\overline{A}; \Gamma}\,[\mathrm{T}?]}{!x(y); P \;|x|\; ?x; Q \vdash \Delta; \Gamma} \qquad \dfrac{\dfrac{\vdots}{R \vdash \Delta, y : \overline{A}; \Gamma, x{:}\overline{A}}}{\mathsf{call}\ x(y); R \vdash \Delta; \Gamma, x{:}\overline{A}}\,[\mathrm{Tcall}]$$

Names in Γ may be used unrestrictedly; each call (typed by [Tcall]) spawns a fresh copy of the server body at type $y : A$, to be used by the client at type $y : \overline{A}$, in a linear binary session. By the typing rule for !A (promotion) such copy does not depend on linear resources. Thus, interaction with replicated objects as captured by the exponentials $!A$ and $?A$ implements a copy semantics where each call obtains a new private *stateless* copy of the same object.

In this work, we introduce a third composition mechanism, allowing processes to concurrently share mutex memory cells, storing *linear state*. Mutex memory cells and their usages are typed respectively by a pair of dual modalities $\mathsf{S}_\bullet A$ and $\mathsf{U}_\bullet A$, whose logical rules are motivated by Differential Linear Logic (DiLL) [34], in particular *cocontraction*, expressed by the type rule [Tsh].

$$\dfrac{P \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma \qquad Q \vdash \Delta', x : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ x\ \{P \mid\mid Q\} \vdash \Delta, \Delta', x : \mathsf{U}_\bullet A; \Gamma}\,[\mathrm{Tsh}]$$

While sharing of replicated objects corresponds to contraction of $?A$ types, shared usage of mutex cells corresponds to cocontraction of $\mathsf{U}_\bullet A$ types. Apart from the explicit use of [Tsh], the type system ensures that memory cells are always used linearly. The shared usage $x : \mathsf{U}_\bullet A$ *is free* in the conclusion of the typing rule, therefore a memory cell may be shared by an arbitrary number of processes, by nested iterated use of cocontraction.

Moreover, cocontraction also ensures that concurrent processes may share a single mutex cell (just like [Tcut] wrt. binary sessions). This constraint comes from the linear logic discipline, and it is important to ensure deadlock freedom. As discussed in Concluding Remarks, this does not hinder CLASS expressiveness - e.g., a single mutex cell may act as a gateway to further bundles of shared state, organised in resource hierarchies, as our examples illustrate - and even suggests convenient concurrent programming structuring techniques.

To access a mutex memory cell in its (unlocked) full state, typed by $\mathsf{U}_\bullet A$, the client uses a *take* operation. Take waits for acquiring the cell lock and reads its contents. The cell then transitions to the (locked) empty state, typed by $\mathsf{U}_\circ A$. The taking client becomes the sole responsible for filling back the cell contents, using a *put* operation. This will restore the cell to the full state, releasing its lock, and making it accessible to other concurrent threads waiting to take it. Our mutex memory cell object is thus akin to a behaviourally typed incarnation of Concurrent Haskell MVars [42] or Rust std::sync::Mutex objects [43].

To ensure safe releasing of a memory cell, its contents are required to be of affine type $\wedge A$. Affine objects are well-behaved disposable values, that when discarded, safely dispose all resources they hereditarily refer to, this being ensured by the linear logic typing.

We illustrate the introduced concepts with a simple example, where two concurrent threads compete to set *on* an initially *off* flag, but only one may win. The flag iteratively announces its state to the client with either #Off or #On. If the state is *off*, the client must select #turnOn, if the state is *on*, it will remain *on*. Process $\mathsf{flag}(f)$ implements the flag (at name f) in the *off* state, and process $\mathsf{on}(f)$ in the *on* state, defined thus

$$\begin{array}{ll} \mathsf{flag}(f) = \#\mathsf{Off}\ f; \mathsf{case}\ f\{\ \mid \#\mathsf{turnOn} : \mathsf{affine}\ f; \mathsf{on}(f)\ \} \\ \mathsf{on}(f) \ \ = \#\mathsf{On}\ f; \mathsf{affine}\ f; \mathsf{on}(f) \end{array}$$

The flag object is typed with the (linear) usage protocol defined by the coinductive type Flag below, such that $\mathsf{flag}(f) \vdash f : \mathsf{Flag}$ and $\mathsf{on}(f) \vdash f : \mathsf{Flag}$

$$\mathsf{type\ corec\ Flag} = \oplus\{\ \mid \#\mathsf{Off} : \&\{\ \mid \#\mathsf{turnOn} : \wedge\mathsf{Flag}\},\ \mid \#\mathsf{On} : \wedge\mathsf{Flag}\}$$

We now consider a scenario where a flag object is shared via a mutex memory cell c initially storing a *off* flag of type $\wedge\mathsf{Flag}$ among two concurrent clients.

```
client(c, id) ⊢ c : U•‾Flag; id : ‾int
client(c, id) =
  take c(f);
  case f {
    |#Off : println id + ": wins.';
            #turnOn f;
            put c(f); release c
    |#On : println id + ": looses.';
            put c(f); release c
  }
```

```
main() ⊢ ∅
main() =
  cut {  cell c(f.affine f; flag(f))
         |c : U•‾Flag|
         share c {
               client(c, 1)
               ||
               client(c, 2)
         }
  }
```

When running $\mathsf{main}()$ exactly one of the threads (executing the same code, just with a different id) will turn the flag *on* and win, the other will loose. Notice that all threads drop usage of the memory cell c using release, which corresponds to DiLL coweakening ([34]).

When considering a new core language, in particular with a static typing discipline, it is necessary to argue about its expressiveness, and aim for a better perception about how natural programs get past its typing rules, and about how types help in structuring programs. In this paper, we approach these concerns by showcasing many interesting examples that challenge the expressiveness of the CLASS language and type system on realistic concurrent programming scenarios. We have developed many more examples, distributed with our implementation, combining imperative, higher-order functional, and session-based programming styles. For all these programs, strong guarantees of memory safety, deadlock-freedom, termination, and absence of "dynamic bugs", even in the presence of blocking primitives and higher-order state, are compositionally certified by our lightweight type discipline based on Propositions-as-Types and Linear Logic.

1.2 Outline and Contributions

We believe that CLASS is the first proposal for a foundational language able to flexibly express realistic concurrent programming idioms while ensuring by typing three key properties: CLASS programs never misuse or leak stateful resources or memory, they never deadlock, and they always terminate.

In Section 2 we formally present the core language CLASS, its type system and operational semantics. Our model builds on the propositions-as-types approach to session-based concurrency [21, 26, 74], extending Second-Order Classical Linear Logic with inductive/coinductive types, affine types, and novel primitives for shareable first-class mutex reference cells for linear state.

In Section 3 we state and prove type preservation (Theorem 3.1), progress (Theorem 3.2) which implies deadlock-freedom, and strong normalisation (Theorem 3.3), which also implies livelock absence. Our proof uses a logical relations argument, extended with an interesting technique to handle shared state interference, which we believe is exploited here for the first time.

Given the strong properties of its type system, it is of course very important to substantiate our claims about CLASS expressiveness. In Section 4 we illustrate the expressiveness of CLASS language and type system by going through a series of compelling examples. Namely, we discuss a general technique for sharing linear protocols, a shareable linked list with update in-place, a shareable buffered channel, using a linked list with pointers to tail and head nodes, and executing send and receive operations in O(1) time; the dining philosophers, illustrating techniques that rely on our type structure to encode resource acquisition hierarchies; a generic barrier for n threads; and a Hoare style monitor with await/notify conditions, where our implementation of the condition's process queue is supported by a dynamic linked data structure, as in real systems code.

Section 5 discusses related work. Section 6 offers concluding remarks and suggests further research. Complete definitions and detailed proofs to all results are provided in the Appendix.

2 The Core Language and its Type System

We present the core language, type system, and operational semantics of CLASS. The language is based on a PaT correspondence with Linear Logic, so terms of the language correspond to proof rules. We start by types and duality.

Definition 2.1 (Types). *Types A, B of* CLASS *are defined by*

$$
\begin{array}{rllllll}
A, B ::= X & \mid \mathbf{1} & \mid \bot & \mid A \mathbin{\&} B & \mid A \oplus B & \mid A \mathbin{⅋} B & \mid A \otimes B \\
 & \mid\, !A & \mid\, ?A & \mid \exists X.A \mid & \mid \forall X.A & \mid \mu X.\ A & \mid \nu X.\ A \\
 & \mid \wedge A & \mid \vee A & \mid \mathsf{S}_{\bullet} A & \mid \mathsf{S}_{\circ} A & \mid \mathsf{U}_{\bullet} A & \mid \mathsf{U}_{\circ} A
\end{array}
$$

Types in the first two rows correspond to propositions of Second-Order Classical Linear Logic, extended with inductive/coinductive types (μ, ν). Types comprise variables (X), units ($\mathbf{1}$, $\bot$), multiplicatives ($\otimes$, ⅋), additives ($\oplus$, $\&$), exponentials (!, ?) and quantifiers ($\exists$, $\forall$). The third row extends this basic type system with affine ($\wedge, \vee$) and new modalities ($\mathsf{S}_{\bullet}, \mathsf{U}_{\bullet}, \mathsf{S}_{\circ}, \mathsf{U}_{\circ}$) to type shared *affine state*.

Duality is the involution operation $A \mapsto \overline{A}$ on types, corresponding to Linear Logic negation, defined by

$$
\begin{array}{lll}
\overline{\mathbf{1}} \;\; = \bot & \overline{A \otimes B} = \overline{A} \mathbin{⅋} \overline{B} & \overline{A \oplus B} = \overline{A} \;\, \& \, \overline{B} \\
\overline{!A} \;= ?\overline{B} & \overline{\exists X.A} \;= \forall X.\overline{A} & \overline{\mu X.\ A} = \nu X.\ \{\overline{X}/X\}(\overline{A}) \\
\overline{\wedge A} = \vee \overline{A} & \overline{\mathsf{S}_{\bullet} A} \;\;\; = \mathsf{U}_{\bullet} \overline{A} & \overline{\mathsf{S}_{\circ} A} \;\;\; = \mathsf{U}_{\circ} \overline{A}
\end{array}
$$

Duality captures symmetry in process interaction, as manifest in the cut rule. In our system, typing judgments have the form $P \vdash_{\eta} \Delta; \Gamma$. The typing context $\Delta; \Gamma$ is dyadic [4, 15, 59, 21], where Δ is handled linearly and Γ is unrestricted; both Δ and Γ assign types to names. The index η is a finite map that holds coinduction hypothesis to type corecursive processes, as detailed later.

Definition 2.2. *The typing rules of* CLASS *are presented in Figs. 1 to 5.*

The type system corresponds, via propositions-as-types [21, 26, 74], to Second-Order Classical Linear Logic (Fig. 1) with inductive/coinductive types (Fig. 2), affinity (Fig. 3) and extended with constructs for shared mutable state (Figs. 4 - 5). The basic composition rules are [Tmix] and [Tcut], which correspond to mix and cut of Linear Logic, respectively. [Tmix] types a parallel composition $P \mid\mid Q$, where P and Q run in parallel without interfering. On the other hand, [Tcut] types linear interactive composition $P \; |x : A| \; Q$: processes P and Q run concurrently and communicate through a private linear session x, session endpoints being typed by dual types $A/\overline{A}$. When the cut type annotation does not play any role, we may omit it and write $P \; |x| \; Q$. In examples, for readibility, we use cut $\{P \; |x| \; Q\}$ and par $\{P \mid\mid Q\}$ instead of $P \; |x| \; Q$ and $P \mid\mid Q$, respectively.

For the basic process constructs [21, 26, 74, 18], $\otimes$/⅋ type send and receive, $\oplus/\&$ type choice and offer (in examples we use labelled choice) , !/? type replicated servers and their invocation, $\forall/\exists$ type receive and send of types, implementing polymorphic processes.

$$\frac{}{\mathbf{0} \vdash_\eta \emptyset; \Gamma}\ [\text{T}\mathbf{0}] \qquad \frac{P \vdash_\eta \Delta'; \Gamma \quad Q \vdash_\eta \Delta; \Gamma}{P \mid\mid Q \ \vdash_\eta \Delta', \Delta; \Gamma}\ [\text{Tmix}]$$

$$\frac{}{\mathsf{fwd}\ x\ y\ \vdash_\eta x : \overline{A}, y : A; \Gamma}\ [\text{Tfwd}] \qquad \frac{P \vdash_\eta \Delta', x : A; \Gamma \quad Q \vdash_\eta \Delta, x : \overline{A}; \Gamma}{P\ |x : A|\ Q\ \vdash_\eta \Delta', \Delta; \Gamma}\ [\text{Tcut}]$$

$$\frac{}{\mathsf{close}\ x \vdash_\eta x : \mathbf{1}; \Gamma}\ [\text{T}\mathbf{1}] \qquad \frac{Q \vdash_\eta \Delta; \Gamma}{\mathsf{wait}\ x; Q\ \vdash_\eta \Delta, x : \bot; \Gamma}\ [\text{T}\bot]$$

$$\frac{P_1 \vdash_\eta \Delta, x : A; \Gamma \quad P_2 \vdash_\eta \Delta, x : B; \Gamma}{\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}\ \vdash_\eta \Delta, x : A \mathbin{\&} B; \Gamma}\ [\text{T}\&]$$

$$\frac{Q_1 \vdash_\eta \Delta', x : A; \Gamma}{x.\mathsf{inl}; Q_1\ \vdash_\eta \Delta', x : A \oplus B; \Gamma}\ [\text{T}\oplus_l] \qquad \frac{Q_2 \vdash_\eta \Delta', x : B; \Gamma}{x.\mathsf{inr}; Q_2\ \vdash_\eta \Delta', x : A \oplus B; \Gamma}\ [\text{T}\oplus_r]$$

$$\frac{P_1 \vdash_\eta \Delta_1, y : A; \Gamma \quad P_2 \vdash_\eta \Delta_2, x : B; \Gamma}{\mathsf{send}\ x(y.P_1); P_2\ \vdash_\eta \Delta_1, \Delta_2, x : A \otimes B; \Gamma}\ [\text{T}\otimes]$$

$$\frac{Q \vdash_\eta \Delta, z : A, x : B; \Gamma}{\mathsf{recv}\ x(z); Q \vdash_\eta \Delta, x : A \parr B; \Gamma}\ [\text{T}\parr]$$

$$\frac{P \vdash_\eta y : A; \Gamma}{!x(y); P\ \vdash_\eta x : !A; \Gamma}\ [\text{T}!] \qquad \frac{Q \vdash_\eta \Delta; \Gamma, x : A}{?x; Q\ \vdash_\eta \Delta, x : ?A; \Gamma}\ [\text{T}?]$$

$$\frac{P \vdash_\eta y : A; \Gamma \quad Q \vdash_\eta \Delta; \Gamma, x : \overline{A}}{y.P\ |!x : A|\ Q\ \vdash_\eta \Delta; \Gamma}\ [\text{Tcut!}] \qquad \frac{Q \vdash_\eta \Delta, z : A; \Gamma, x : A}{\mathsf{call}\ x(z); Q\ \vdash_\eta \Delta; \Gamma, x : A}\ [\text{Tcall}]$$

$$\frac{P \vdash_\eta \Delta, x : \{B/X\}A; \Gamma}{\mathsf{sendty}\ x(B); P\ \vdash_\eta \Delta, x : \exists X.A; \Gamma}\ [\text{T}\exists] \qquad \frac{Q \vdash_\eta \Delta, x : A; \Gamma}{\mathsf{recvty}\ x(X); Q \vdash_\eta \Delta, x : \forall X.A; \Gamma}\ [\text{T}\forall]$$

Fig. 1: Typing Rules I: Second-Order CLL.

$$\frac{P \vdash_{\eta'} \Delta, z : A; \Gamma \quad \eta' = \eta, X(z, \vec{w}) \mapsto \Delta, z : Y; \Gamma}{\mathsf{corec}\ X(z, \vec{w}); P\ [x, \vec{y}] \vdash_\eta \{\vec{y}/\vec{w}\}\Delta, x : \nu Y.\ A; \{\vec{y}/\vec{w}\}\Gamma}\ [\text{Tcorec}]$$

$$\frac{\eta = \eta', X(x, \vec{y})\ \mapsto \Delta, x : Y; \Gamma}{X(z, \vec{w}) \vdash_\eta \{\vec{w}/\vec{y}\}\Delta, z : Y; \{\vec{w}/\vec{y}\}\Gamma}\ [\text{Tvar}]$$

$$\frac{P \vdash_\eta \Delta, x : \{\mu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\mu\ x; P \vdash_\eta \Delta, x : \mu X.\ A; \Gamma}\ [\text{T}\mu] \qquad \frac{P \vdash_\eta \Delta, x : \{\nu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\nu\ x; P \vdash_\eta \Delta, x : \nu X.\ A; \Gamma}\ [\text{T}\nu]$$

Fig. 2: Typing Rules II: Induction and Coinduction.

$$\frac{P \vdash_\eta a : A, \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}; \Gamma}{\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ \vdash_\eta a : \wedge A, \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}; \Gamma}\ [\text{Taffine}]$$

$$\frac{}{\mathsf{discard}\ a \vdash_\eta a : \vee A; \Gamma}\ [\text{Tdiscard}] \qquad \frac{Q \vdash_\eta \Delta, a : A; \Gamma}{\mathsf{use}\ a; Q \vdash_\eta \Delta, a : \vee A; \Gamma}\ [\text{Tuse}]$$

Fig. 3: Typing Rules III: Affinity.

Coinductive types are introduced by rule [Tcorec]. It types corecursive processes $\mathsf{corec}\ X(z, \vec{w}); P\ [x, \vec{y}]$, with parameters $z, \vec{w}$ bound in P, that are instantiated with the arguments $x, \vec{y}$ (free in the process term). By convention, the coinductive behaviour, of type $\nu Y.\ A$, of a corecursive process is always offered in the first argument z. According to [Tcorec], to type the body P of a corecursive process, the map η is extended with a coinductive hypothesis binding the process variable X to the typing context $\Delta, z : Y; \Gamma$, so that when typing the body P of the corecursion we can appeal to X, which intuitively stands for P itself, and recover its typing invariant. Crucially, the type variable Y is free only in $z : A$. This causes corecursive calls to be always applied to names z' that hereditarily descend from the initial corecursive argument z, a necessary condition for strong normalisation (Theorem 3.3), and morally corresponds to only allowing corecursive calls on "smaller" argument sessions (of inductive type).

Rule [Tvar] types a corecursive call $X(z, \vec{w})$ by looking up in η for the corresponding binding and renaming the parameters with the arguments of the call. Inductive and coinductive types are explicitly unfolded with [Tμ] and [Tν].

To simplify the presentation in program examples, we omit explicit unfolding actions, and write inductive and coinductive type definitions with equations of the form $\mathsf{rec}\ A = f(A)$ and $\mathsf{corec}\ B = f(B)$ instead of $A = \mu X.\ f(X)$ and $B = \nu X.\ f(X)$, respectively. Similarly, we write corecursive process definitions as $Q(x, \vec{y}) = f(Q(-))$ instead of $Q(x, \vec{y}) = \mathsf{corec}\ X(z, \vec{w}); f(X(-))\ [x, \vec{y}]$, while of course respecting the constraints imposed by typing rules [Tvar] and [Tcorec].

Affinity Affinity is important to model discardable linear resources, and plays an important role in CLASS. An affine session can either be used as a linear session or discarded. The typing rules for the affine modalities are in Fig. 3. Affine sessions are introduced by rule [Taffine] that promotes a linear $a : A$ to an affine session $a : \wedge A$. It types $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P$, which provides an affine session at a and continues as P, and follows the structure of a standard promotion rule.

A session a may be promoted to affine if it only depends on resources that can be disposed, i.e. resources that satisfy some form of weakening capability, namely: coaffine sessions b_i of type $\vee B_i$, that can be discarded; full cell usages c_i of type with $\mathsf{U}_\bullet C_i$, that can be released; and unrestricted sessions in Γ, which are implicitly ?-typed. The dependencies of an affine object on coaffine or full cell objects are explicitly annotated $\vec{b}, \vec{c}$ in the process term, to instrument the operational semantics, but we often omit them and simply write $\mathsf{affine}\ a; P$.

$$\frac{P \vdash_\eta \Delta, a : \wedge A; \Gamma}{\mathsf{cell}\ c(a.P) \vdash_\eta \Delta, c : \mathsf{S}_\bullet A; \Gamma}\ [\text{Tcell}] \qquad \frac{}{\mathsf{release}\ c \vdash_\eta c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Trelease}]$$

$$\frac{}{\mathsf{empty}\ c \vdash_\eta c : \mathsf{S}_\circ A; \Gamma}\ [\text{Tempty}] \qquad \frac{Q \vdash_\eta \Delta, a : \vee A, c : \mathsf{U}_\circ A; \Gamma}{\mathsf{take}\ c(a); Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Ttake}]$$

$$\frac{Q_1 \vdash_\eta \Delta_1, a : \wedge \overline{A}; \Gamma \qquad Q_2 \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{put}\ c(a.Q_1); Q_2\ \vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\circ A; \Gamma}\ [\text{Tput}]$$

Fig. 4: Typing Rules IV: Reference Cells.

The coaffine endpoint $\vee A$ of an affine session, dual of $\wedge \overline{A}$, has two operations: use and discard. Rule [Tuse] types a process $\mathsf{use}\ a; Q$ that uses a coaffine session a and continues as Q, it is a dereliction rule. [Tdiscard] types the process $\mathsf{discard}\ a$ that discards a coaffine session a, it is a weakening rule.

Shared Mutable State Shared state is introduced in CLASS by typed constructs that model mutex memory cells, and associated cell operations allowing its use by client code, defined by the tying rules in Fig. 4.

At any moment a cell may be either *full* or *empty*, akin to the MVars of Concurrent Haskel [42]. A full cell on c, written $\mathsf{cell}\ c(a.P)$, is typed $\mathsf{S}_\bullet A$ by rule [Tcell]. Such cell stores an *affine* session of type $\wedge A$, implemented at a by P. All objects stored in cells are required to be affine, so that memory cells may always be safely disposed without causing memory leaks. An empty cell on c, of type $\mathsf{S}_\circ A$, and written $\mathsf{empty}\ c$, is typed by rule [Tempty].

Client processes manipulate cells via *take*, *put* and *release* operations. These operations apply to names of cell usage types - $\mathsf{U}_\bullet A$ (full cell usage) and $\mathsf{U}_\circ A$ (empty cell usage) - which are dual types of $\mathsf{S}_\bullet \overline{A}$ and $\mathsf{S}_\circ \overline{A}$, respectively. At any given moment, a client thread owning a $\mathsf{U}_\bullet A$-typed usage to a cell may execute a *take* operation, typed by rule [Ttake]. The *take* operation $\mathsf{take}\ c(a); Q$ waits to acquire the cell mutex c, and reads its contents into parameter a, the linear (actually coaffine, of type $\vee A$) usage for the object stored in the cell; the cell becomes empty, and execution continues as Q.

It is responsibility of the taking thread to put some value back in the empty cell, thus releasing the lock, causing the cell to transition to the full state. The *put* operation $\mathsf{put}\ c(a.Q_1); Q_2$ is typed by [Tput], the stored object a, implemented by Q_1, is required to be affine, as specified in the premise $a : \wedge \overline{A}$.

Hence a cell flips from full to empty and back; [Ttake] uses the cell c at $\mathsf{U}_\bullet A$ type, and its continuation (in the premise) at $\mathsf{U}_\circ A$ type, symmetrically [Tput] uses the cell c at $\mathsf{U}_\circ A$ type, and its continuation (in the premise) at $\mathsf{U}_\bullet A$ type.

The $\mathsf{release}\ c$ operation allows a thread to manifestly drop its cell usage c. Release is typed by [Trelease] (cf. coweakening [34]); a usage may only be released in the unlocked state $\mathsf{U}_\bullet A$. When, for some cell c, all the owning threads release

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\bullet A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Tsh}]$$

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\circ A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\circ A; \Gamma}\ [\text{TshL}]$$

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\bullet A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\circ A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\circ A; \Gamma}\ [\text{TshR}]$$

Fig. 5: Typing Rules V: State Sharing.

their usages, which eventually happens in well-typed programs, the cell c gets disposed, and its (affine) contents safely discarded.

Our memory cells cells are linear objects, with a linear mutable payload, which are never duplicated by reduction or conversion rules. However, in CLASS, multiple cell usages may be shared between concurrent threads, which compete to take and use it in interleaved critical sections. Such aliased usages be passed around and duplicated dynamically, changing the sharing topology at runtime.

Sharing of cell usages is logically expressed in our system by the typing rules in Fig. 5. Co-contraction, introduced in Differential Linear Logic DiLL [34], allows finite multisets of linear resources to safely interact in cut-reduction, resolving concurrent sharing into nondeterminism, as required here to soundly model memory cells and their linear concurrent usages. Rule [Tsh] interprets cocontraction with the construct $\mathsf{share}\ c\ \{P \mid\mid Q\}$, and types sharing of the cell usage $c : \mathsf{U}_\bullet A$ between the concurrent threads P and Q.

Contrary to cut, $\mathsf{share}\ c\ \{P \mid\mid Q\}$ is *not* a binding operator for c. The shared usage $c : \mathsf{U}_\bullet A$ is *free* in the conclusion of the typing rule, permitting c to be shared among an arbitrary number of threads, by nested iterated use of [Tsh]. In [Tsh], P and Q only share the single mutex cell c, since the linear context is split multiplicatively, just like [Tcut] wrt. binary sessions. This condition comes from the DiLL typing discipline, and is important to ensure deadlock freedom.

While [Tsh] types sharing of a full (unlocked) cell usage of type $\mathsf{U}_\bullet A$, the symmetric rules [TshR] and [TshR] type sharing of an empty (locked) cell usage of type $\mathsf{U}_\circ A$. We may verify that for every cell c in a well-typed process, at most one unguarded operation to c may be using type $\mathsf{U}_\circ A$, all the remaining unguarded operations to c must be using type $\mathsf{U}_\bullet A$. This implies that, at runtime, only one thread may own the lock for a given (necessarily empty) cell, and execute a *put* to it, which will bring the cell back to full and release its lock, other threads must be either attempting to *take*, or *release* the reference.

Working together, the sharing typing rules ensure that in any well-typed cell sharing tree, at most one single thread at any time may be actively using a cell (in the locked empty state) and put to it, thus guaranteeing mutual exclusion, while satisfying Progress (Theorem 3.2) which in turn ensures deadlock absence, even in the presence of the crucially blocking behaviour of the take operation.

$$
\begin{array}{lr}
\mathsf{fwd}\ x\ y \equiv \mathsf{fwd}\ y\ x \quad P\ |x|\ Q \equiv Q\ |x|\ P & \\
\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv \mathsf{share}\ x\ \{Q \mid\mid P\} & [\text{comm}] \\
P \mid\mid \mathsf{0} \equiv P \quad P \mid\mid Q \equiv Q \mid\mid P \quad P \mid\mid (Q \mid\mid R) \equiv (P \mid\mid Q) \mid\mid R & [\text{par}] \\
P\ |x|\ (Q \mid\mid R) \equiv (P\ |x|\ Q) \mid\mid R & [\text{CM}] \\
P\ |x|\ (Q\ |y|\ R) \equiv (P\ |x|\ Q)\ |y|\ R & [\text{CC}] \\
P\ |x|\ \mathsf{share}\ y\ \{Q \mid\mid R\} \equiv \mathsf{share}\ y\ \{P\ |x|\ Q \mid\mid R\} & [\text{CSh}] \\
P\ |z|\ (y.Q\ |!x|\ R) \equiv y.Q\ |!x|\ (P\ |z|\ R) & [\text{CC!}] \\
y.Q\ |!x|\ (P \mid\mid R) \equiv P \mid\mid (y.Q\ |!x|\ R) & [\text{C!M}] \\
y.P\ |!x : A|\ (w.Q\ |!z : B|\ R) \equiv w.Q\ |!z : B|\ (y.P\ |!x : A|\ R) & [\text{C!C!}] \\
\mathsf{share}\ x\ \{P \mid\mid (Q \mid\mid R)\} \equiv \mathsf{share}\ x\ \{P \mid\mid Q\} \mid\mid R & [\text{ShM}] \\
\mathsf{share}\ x\ \{P \mid\mid \mathsf{share}\ y\ \{Q \mid\mid R\}\} \equiv \mathsf{share}\ y\ \{\mathsf{share}\ x\ \{P \mid\mid Q\} \mid\mid R\} & [\text{ShSh}] \\
\mathsf{share}\ z\ \{P \mid\mid y.Q\ |!x|\ R\} \equiv y.Q\ |!x|\ \mathsf{share}\ z\ \{P \mid\mid R\} & [\text{ShC!}] \\
y.P\ |!x : A|\ (Q * R) \equiv (y.P\ |!x : A|\ Q) * (y.P\ |!x : A|\ R) & [\text{D-C!X}] \\
\mathsf{share}\ x\ \{\mathsf{release}\ x \mid\mid P\} \leq P & [\text{ShRel}] \\
\mathsf{share}\ x\ \{\mathsf{put}\ x(y.P); Q \mid\mid R\} \leq \mathsf{put}\ x(y.P); \mathsf{share}\ x\ \{Q \mid\mid R\} & [\text{ShPut}] \\
\mathsf{share}\ x\ \{\mathsf{take}\ x(y_1); P_1 \mid\mid \mathsf{take}\ x(y_2); P_2\} & \\
\quad \leq \mathsf{take}\ x(y_1); \mathsf{share}\ x\ \{P_1 \mid\mid \mathsf{take}\ x(y_2); P_2\} & [\text{ShTake}]
\end{array}
$$

Provisos: in [CM] and [ShM], $x \in \mathsf{fn}(Q)$; in [CC], [CSh] and [ShSh], $x, y \in \mathsf{fn}(Q)$; in [CC!], [C!M] and [ShC!], $x \notin \mathsf{fn}(P)$; in [C!C!], $x \notin \mathsf{fn}(Q)$ and $z \notin \mathsf{fn}(P)$.

Fig. 6: Structural congruence $P \equiv Q$ and precongruence $P \leq Q$.

2.1 Operational Semantics

We now define CLASS operational semantics, which is given by a structural precongruence relation $\leq$ that captures static relations on processes, essentially rearranging them, and a reduction relation $\rightarrow$ that captures process interaction.

Definition 2.3 ($P \equiv Q$ and $P \leq Q$). *Structural congruence $\equiv$ is the least congruence on processes closed under α-conversion and the $\equiv$-rules in Fig. 6. Structural precongruence $\leq$ is the least precongruence on processes including $\equiv$ and closed under α-conversion and the $\leq$-rules in Fig. 6.*

The basic rules of $\equiv$ essentially reflect the expected static laws, along the lines of the structural congruences / conversions in [21, 74]. The binary operators forwarder, cut and share are commutative ([comm]). The set of processes modulo $\equiv$ is a commutative monoid with binary operation given by parallel composition and identity given by inaction $\mathsf{0}$ ([par]). Any two static constructs commute, as expressed by the laws [CM]-[ShC!]. Furthermore, we can distribute the unrestricted cut over all the static constructs as expressed by law [D-C!X], where $*$ stands for either a mix, linear or unrestricted cut or a share.

The commuting conversions [ShTake] and [ShPut] allows take and put operations on cell usages to commute with a share construct. Rule [ShTake] picks the take that occurs on the left argument, however since share is commutative, a right-biased version of [ShTake] is admissible. Using [ShTake], any of the two possible interleavings for two concurrent takes may be nondeterministically picked via $\leq$. Indeed, we express $\leq$ as a precongruence because it introduces nondeterminism, and does not express a behavioural equivalence as $\equiv$ does. N.B.: Although one could easily formulate a confluent version of CLASS semantics, using explicit sums as in [13, 61, 34], we prefer in this paper to focus on the expressiveness of CLASS as a programming language and on its deadlock and livelock absence properties, adopting a nondeterministic reduction relation.

In [ShPut] only a put, in the $\mathsf{U}_\circ A$-typed premise of [TshL], may be propagated up and eventually update the cell, causing it to transit back to the full state. Hence, take operations originating the $\mathsf{U}_\bullet A$ typed premise of [TshR] will be blocked, waiting until such (unique) put propagation occurs. Algebraically, rule [ShRel] expresses that the release operation is the identity for share composition, we orient it as a precongruence, to ensure type preservation.

Definition 2.4 (Reduction $\rightarrow$). *Reduction $\rightarrow$ is defined by the rules of Fig. 7.*

We let $\xrightarrow{*}$ stand for the reflexive-transitive closure of $\rightarrow$. Reduction includes the set of principal cut conversions, i.e. the redexes for each pair of interacting constructs. It is closed by structural precongruence ([$\leq$]) and in rule [cong] we consider that $\mathcal{C}$ is a static context, i.e. a process context in which the hole is covered only by the static constructs mix, cut and share.

Operationally, the forwarding behaviour is implemented by name substitution [22] ([fwd]). All the other conversions apply to a principal cut between two dual actions. Reduction rules for the basic session constructs that interpret Second Order Linear Logic and recursion are the expected ones [21, 26, 75], along predictable lines. For readability, we omit the type declarations in the cuts, as they do not actually play any role in reduction.

We comment the rules concerning affinity. The interaction between an affine session and an use operation is defined by reduction rule [$\wedge\vee$u], where a cut on $a : \wedge A$ between $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P$ and $\mathsf{use}\ a; Q$ reduces to a cut on $a : A$ between the continuations P and Q. The reduction between an affine session and a discard operation is defined by [$\wedge\vee$d]. A cut between $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P$ and $\mathsf{discard}\ a$ reduces to a mix-composition of discards (for the coaffine sessions $\vec{b}$) and releases (for the cell usages $\vec{c}$) cf. [6, 19]). In the corner case where $\vec{c}$ and $\vec{a}$ are empty, the left-hand side of [$\wedge\vee$d] simply degenerates to inaction $\mathbf{0}$ (the identity of mix).

The reductions for the mutable state operations are fairly self-explanatory. In rule [$\mathsf{S}_\bullet\mathsf{U}_\bullet$r], a cut between a full mutex cell cell and a release operation reduces to a process that discards the affine cell contents, cf. rule [$\wedge\vee$d]. In rule [$\mathsf{S}_\bullet\mathsf{U}_\bullet$t], a cut on $c : \mathsf{S}_\bullet A$ between a full cell and a take operation reduces to a process with two cuts, both composed with the continuation $\{a/a'\}Q$ of the take. The outer cut on $a : \wedge A$ composes with the stored affine session, which was successfully acquired by the take operation. The inner cut on $c : \mathsf{S}_\circ A$ composes with the

$$
\begin{array}{ll}
\mathsf{fwd}\ x\ y\ |y|\ P \to \{x/y\}P & [\text{fwd}]\\
\mathsf{close}\ x\ |x|\ \mathsf{wait}\ x; P \to P & [\mathbf{1}\bot]\\
\mathsf{send}\ x(y.P); Q\ |x|\ \mathsf{recv}\ x(z); R \to Q\ |x|\ (P\ |y|\ \{y/z\}R) & [\otimes⅋]\\
\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x|\ x.\mathsf{inl}; R \to P\ |x|\ R & [\&\oplus_l]\\
\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x|\ x.\mathsf{inr}; R \to Q\ |x|\ R & [\&\oplus_r]\\
!x(y); P\ |x|\ ?x; Q \to y.P\ |!x|\ Q & [!?]\\
y.P\ |!x|\ \mathsf{call}\ x(z); Q \to \{z/y\}P\ |z|\ (y.P\ |!x|\ Q) & [\text{call}]\\
\mathsf{sendty}\ x(A); P\ |x|\ \mathsf{recvty}\ x(X); Q \to P\ |x|\ \{A/X\}Q & [\exists\forall]\\
\mathsf{unfold}_\mu\ x; P\ |x|\ \mathsf{unfold}_\nu\ x; Q \to P\ |x|\ Q & [\mu\nu]\\
\mathsf{unfold}_\mu\ x; P\ |x|\ \mathsf{corec}\ Y(z, \vec{w}); Q\ [x, \vec{y}] & \\
\quad \to P\ |x|\ \{x/z\}\{\vec{y}/\vec{w}\}\{\mathsf{corec}\ Y(z, \vec{w}); Q/Y\}Q & [\text{corec}]\\
\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ |a|\ \mathsf{use}\ a; Q \to P\ |a|\ Q & [\wedge\vee\text{u}]\\
\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ |a|\ \mathsf{discard}\ a \to \mathsf{discard}\ \vec{b}\ ||\ \mathsf{release}\ \vec{c} & [\wedge\vee\text{d}]\\
\mathsf{cell}\ c(a.P)\ |c|\ \mathsf{release}\ c \to P\ |a|\ \mathsf{discard}\ a & [\mathsf{S}_\bullet\mathsf{U}_\bullet\text{r}]\\
\mathsf{cell}\ c(a.P)\ |c|\ \mathsf{take}\ c(a'); Q \to P\ |a|\ (\mathsf{empty}\ c\ |c|\ \{a/a'\}Q) & [\mathsf{S}_\bullet\mathsf{U}_\bullet\text{t}]\\
\mathsf{empty}\ c\ |c|\ \mathsf{put}\ c(a.P); Q \to \mathsf{cell}\ c(a.P)\ |c|\ Q & [\mathsf{S}_\circ\mathsf{U}_\circ]\\
P \le P' \text{ and } P' \to Q' \text{ and } Q' \le Q\ \supset\ P \to Q & [\le]\\
P \to Q\ \supset\ \mathcal{C}[P]\ \to \mathcal{C}[Q] & [\text{cong}]
\end{array}
$$

Fig. 7: Reduction $P \to Q$.

reference cell c, which has became empty in the reductum. Finally, in rule $[\mathsf{S}_\circ\mathsf{U}_\circ]$, a cut on session $c : \mathsf{S}_\circ A$ between an empty cell and a put operation reduces to a cut on session $c : \mathsf{S}_\bullet A$ between a full cell, that now stores the session that was put, and the continuation of the put process. Notice that the locking/unlocking behaviour of cells is simply modelled by rewriting of the process terms, from cell to empty and back, as typical in process calculi.

3 Type Safety and Strong Normalisation

In this section we state and give proof sketches for our main results of type safety and strong normalistion. Full proofs may be found in the Appendix.

Type Preservation The semantics of CLASS is defined by a set of precongruence $\le$ and reduction $\to$ rules on process terms. Theorem 3.1 shows that these relations preserve typing, and gives substance to our PaT approach, showing that every $\le$ and $\to$ rule corresponds to a conversion on type derivations/proofs.

Theorem 3.1 (Type Preservation). *Suppose $P \vdash_\eta \Delta; \Gamma$. (1) If $P \le Q$, then $Q \vdash_\eta \Delta; \Gamma$. (2) If $P \to Q$, then $Q \vdash_\eta \Delta; \Gamma$.*

Proof. By induction on derivations for $P \le Q$ (resp. $P \to Q$), we verify that all the rules of $\le$ (Def. 2.3) (resp. $\to$ (Def. 2.4)) are type preserving.

Progress We prove the progress property for well-typed CLASS processes. The following notion of *live process* becomes useful. A process P is *live* if and only if $P = \mathcal{C}[Q]$, for some static context $\mathcal{C}$ (the hole lies within the scope of static constructs mix, cut and share) and Q is an active process (a process with a topmost action prefix, such as a receive or a take, or a forwarder). We first show that a live well-typed process either reduces or offers an interaction with its environment on a free name. The following observability predicate (cf. [64]) characterises the interactions of a process with its environment

Definition 3.1 **($P \downarrow_x$).** *The predicate $P \downarrow_x$ is defined by rules of Fig. 8.*

The predicate $P \downarrow_x$ holds if P offers an immediate interaction (unguarded action) on free name x. We can observe the subject of an action (rule [act]) and x, y of a forwarder $\mathsf{fwd}\ x\ y$. The definition of $P \downarrow_x$ is closed by $\leq$ and propagates observations over the various static operators. Cut bound names are not free, hence cannot be observed. Share $\mathsf{share}\ y\ \{P \mid\mid Q\}$ propagates all the observations x for which $x \neq y$ and by applying $\leq$ rules [ShTake], [ShRel] or [ShPut] via [$\leq$], an interaction on x may be observed. We have

Lemma 3.1 **(Liveness).** *Let $P \vdash_\emptyset \Delta; \Gamma$ be live. Either $P \downarrow_x$ or P reduces.*

Proof. (Sketch) By induction on a derivation for $P \vdash_\emptyset \Delta; \Gamma$, along the lines of [26]. To handle case [Tcut] $P = P_1 \mid y \mid P_2$: both P_1 and P_2 are live, since both type with a nonempty linear typing context, hence we can apply the induction hypothesis (i.h.) to both premises of [Tcut]: either (i) one of P_1 and P_2 reduces or (ii) both $P_1 \downarrow_{x_1}$ and $P_2 \downarrow_{x_2}$. If (i), then P reduces. Case (ii) follows because, crucially, P_1 and P_2 synchronise through a single private session y, then either $x_1 \neq y$ or $x_2 \neq y$, in which case we can observe either x_1 or x_2; or $x_1 = x_2 = y$, in which case we can trigger a reduction, by applying $\leq$ rules to P in order to exhibit a principal cut. For case [Tsh] $P = \mathsf{share}\ y\ \{P_1 \mid\mid P_2\}$: since P_1 and P_2 are live, we apply i.h. to both premises. The interesting case occurs when $P_1 \downarrow_{x_1}$ and $P_2 \downarrow_{x_2}$. Co-contraction implies that P_1 and P_2 share the single usage y, so if $x_1 \neq y$ or $x_2 \neq y$, we have either $P_1 \downarrow_{x_1}$ or $P_1 \downarrow_{x_2}$. If both $x_1 = x_2 = y$, then we derive $P \downarrow_y$: the observation corresponds to either a take or a release operation on y, which we commute up with [ShTake] or [ShRel]. For [TshL] $P = \mathsf{share}\ y\ \{P_1 \mid\mid P_2\}$, we apply the i.h. to the premise P_1, whcih types with an empty usage on y. If $P_1 \downarrow_y$, then $P \downarrow_y$, the observation corresponding a put operation on y, which we commute up with [ShPut]. Symmetrically for [TshR].

Theorem 3.2 **(Progress).** *Let $P \vdash_\emptyset \emptyset; \emptyset$ be a live process. Then, P reduces.*

Proof. Follows from Lemma 3.1 since $\mathsf{fn}(P) = \emptyset$.

Remarkably, our proof of Theorem 3.2 leverages deep properties of Linear Logic, in particular the structure of the linear cut and co-contraction, allowing us to prove deadlock absence, even in a language with primitives exhibiting blocking behaviour, avoiding the use of extra mechanisms [44, 32, 45, 10, 24, 70, 30].

$$\frac{}{\mathsf{fwd}\ x\ y \downarrow_x}\,[\text{fwd}] \quad \frac{s(\mathcal{A}) = x}{\mathcal{A} \downarrow_x}\,[\mathcal{A}] \quad \frac{P \leq Q \quad Q \downarrow_x}{P \downarrow_x}\,[\leq] \quad \frac{P \downarrow_x}{(P \mid\mid Q) \downarrow_x}\,[\text{mix}]$$

$$\frac{P \downarrow_x \quad x \neq y}{(P\ |y|\ Q) \downarrow_x}\,[\text{cut}] \quad \frac{Q \downarrow_x \quad x \neq y}{(z.P\ |!y|\ Q) \downarrow_x}\,[\text{cut!}] \quad \frac{P \downarrow_x \quad x \neq y}{(\mathsf{share}\ y\ \{P \mid\mid Q\}) \downarrow_x}\,[\text{share}]$$

Fig. 8: Observability Predicate $P \downarrow_x$.

Strong Normalisation Establishing strong normalisation (SN) for concurrent process calculi is usually fairly challenging, particularly in the presence of name passing, recursion and higher-order shared state [31, 16, 77, 46, 63]. For example, with reference cells one may express general recursion with Landin's knot, and, in general, circular chains of references that may lead to divergence. However, our linear type system uses primitive recursion and corecursion, and excludes cyclic dependencies through state or session based interaction, allowing strong normalisation, and therefore livelock absence, to hold.

Our proof relies on defining suitable linear logical relations, cf. [58, 20, 66], adapted to Classical Linear Logic [37, 1, 8], and crucially relying on a notion of reducibility up to interference that imposes stronger properties on the interpretation of the state modalities, and which allows the inductive proof of the Fundamental Lemma 3.2 to go through in the usual way. To this end, we extend our basic language with auxiliary constructs $\mathsf{cell}\ c(a.S)$ and $\mathsf{empty}\ c(a.S)$, which denote memory cells subject to interference from concurrent writers, allowed to take terms from the set $S \subseteq \{P \mid P \vdash_\eta a : \wedge A\}$. The intuition is that a take on the cell may always read any object from S, due to interference. We also consider the following additional reduction (nondeterministic) rules (1)-(3), where in 1 and 2 we assume $P \in S$.

$$\begin{array}{lll}\mathsf{cell}\ c(a.S)\ |c|\ \mathsf{release}\ c & \to P\ |a|\ \mathsf{discard}\ a, & (1)\\ \mathsf{cell}\ c(a.S)\ |c|\ \mathsf{take}\ c(a'); Q & \to \mathsf{empty}\ c(a.S)\ |c|\ (P\ |a|\ \{a/a'\}Q) & (2)\\ \mathsf{empty}\ c(a.S)\ |c|\ \mathsf{put}\ c(a.P); Q & \to \mathsf{cell}\ c(a.S)\ |c|\ Q & (3)\end{array}$$

In this section, we thus consider reduction of $P \to Q$ to be the relation defined in Fig 7, extended with these rules. When a take or a release interacts with $\mathsf{cell}\ c(a.S)$, an arbitrary element P from the set S may be picked (rules (1) and (2)). In (3), a put $\mathsf{put}\ c(a.P); Q$ interacts with $\mathsf{empty}\ c(a.S)$ causing $\mathsf{empty}\ c(a.S)$ to evolve to $\mathsf{cell}\ c(a.S)$ (3). The following notion is also useful. A process P is *S-preserving on x* if $P \vdash_\eta x : \mathsf{U}_\bullet A$ or $P \vdash_\eta x : \mathsf{U}_\circ A$, and

- if $P \xrightarrow{*}\approx \mathsf{take}\ x(y); P'$ and $Q \in S$, then $Q\ |y|\ P'$ is S-preserving on x.
- if $P \xrightarrow{*}\approx \mathsf{put}\ x(y.P_1); P_2$, then $P_1 \in S$ and P_2 is S-preserving on x.

A set of processes T is S-preserving on x if and only for all $P \in T$, P is S-preserving on x. Intuitively a process P that uses a cell x is S-preserving on x if it only puts values from S on cell x. The notion of S-preservation, parametric on any S, brings explicit the conditions needed for safe interaction with a memory cell, subject to interference, while ensuring a state invariant S on the cell contents. We now introduce the logical predicate.

Definition 3.2 (Logical Predicate $[\![x : A]\!]_\sigma$). *By induction on the type A, we define the sets $[\![x : A]\!]_\sigma$ an shown in Fig. 9, such that $[\![x : \mathsf{U}_\bullet A]\!]_\sigma$ and $[\![x : \mathsf{U}_\circ A]\!]_\sigma$ are $[\![- : \wedge\overline{A}]\!]$-preserving on x.The definition is direct for the positive types A, for negative types B is given by orthogonality.*

The definition relies on Girard's notion of orthogonality $S^\perp \triangleq \{P \mid \forall Q \in S.\ P\ |x|\ Q \text{ is SN}\}$ [36]. Duality promotes succinctness in our definition: for negative types A, $[\![x : A]\!]_\sigma$ is defined as the orthogonal of the predicate for its dual $\overline{A}$ (positive) type. To handle polymorphic and inductive types, the logical predicate is indexed by a map σ that assigns reducibility candidates $R[x : A]$ to type variables. A reducibility candidate $R[x : A]$ is any set S of processes $P \vdash_\emptyset x : A$ such that P is SN and $S = S^{\perp\perp}$. We let $\mathcal{R}[- : A]$ be the set of all reducibility candidates $R[x : A]$ for some name x. The definition relies on a congruence relation $\approx$ extending $\leq$ with a complete set of commuting conversions, along standard lines [21, 26, 74]. It essentially plays the role of the labelled transition system in the proof of strong normalisation given in [58].

We now extend the logical predicate to typing judgements $P \vdash_\eta \Delta; \Gamma$ by universal closure over the typing context and σ.

Definition 3.3 (Extended Logical Predicate $\mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$). *We define $\mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$ inductively on Δ, Γ and η as the set of processes $P \vdash_\eta \Delta; \Gamma$ s.t.*

$P \in \mathcal{L}[\![\vdash_\emptyset \emptyset; \emptyset]\!]_\sigma$ *iff* P *is SN.*
$P \in \mathcal{L}[\![\vdash_\emptyset \Delta, x : A; \Gamma]\!]_\sigma$ *iff* $\forall Q \in [\![x : \overline{A}]\!]_\sigma.\ Q\ |x : \overline{A}|\ P \in \mathcal{L}[\![\vdash_\emptyset \Delta; \Gamma]\!]_\sigma$.
$P \in \mathcal{L}[\![\vdash_\emptyset \Delta; \Gamma, x : A]\!]_\sigma$ *iff* $\forall Q \in [\![y : \overline{A}]\!]_\sigma.\ y.Q\ |!x : \overline{A}|\ P \in \mathcal{L}[\![\vdash_\emptyset \Delta; \Gamma]\!]_\sigma$.
$P \in \mathcal{L}[\![\vdash_{\eta, X(x,\vec{y}) \mapsto \Delta', x:Y;\Gamma} \Delta; \Gamma]\!]_\sigma$ *iff* $\forall Q \in \sigma(Y).\ \{Q/X\}P \ \in \mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$.

We now state the Fundamental Lemma (3.2) from which Theorem 3.3 follows.

Lemma 3.2 (Fundamental Lemma). *If $P \vdash_\eta \Delta; \Gamma$, then $P \in \mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$.*

Proof. (Sketch) By induction on $P \vdash_\eta \Delta; \Gamma$. To handle cases [Tcell] and [Tempty], we show that $\mathsf{cell}\ c(a.S)$ and $\mathsf{empty}\ c(a.S)$ respectively simulate $\mathsf{cell}\ c(a.P)$ (where $P \in S$) and $\mathsf{empty}\ c$, when composed with any S-preserving on c usages. To handle one of the most challenging cases, [Tsh] we prove, for all S, and all S-preserving on x processes P_1 and P_2, that $\mathsf{cell}\ c(a.S)\ |c|\ \mathsf{share}\ c\ \{P_1 \mid\mid P_2\}$ (1) is simulated by $(\mathsf{cell}\ c(a.S)\ |c|\ P_1) \mid\mid (\mathsf{cell}\ c(a.S)\ |c|\ P_2)$ (2). This allows us to infer that if (2) is SN, then so it is (1). When $S = [\![a : \wedge\overline{A}]\!]_\sigma$, the i.h. yields $(\mathsf{cell}\ c(a.S)\ |c|\ P_i)$ SN, hence we conclude (2) SN. Similarly for [TshL], [TshR].

Theorem 3.3 (Strong Normalisation). *If $P \vdash_\emptyset \emptyset; \emptyset$, then P is SN.*

4 Typeful Concurrent Programming in CLASS

In this section, we discuss the expressiveness of CLASS language and type system, by going through a sequence of illustrative and realistic concurrent programming idioms, all of which are validated by our implementation.

$$
\begin{aligned}
[\![x : X]\!]_\sigma &\triangleq \sigma(X)[x] \\
[\![x : \mathbf{1}]\!]_\sigma &\triangleq \{P \mid P \approx \mathsf{close}\ x \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : A \otimes B]\!]_\sigma &\triangleq \{P \mid \exists P_1, P_2.\ P \approx \mathsf{send}\ x(y.P_1); P_2 \text{ and} \\
&\qquad P_1 \in [\![y : A]\!]_\sigma \text{ and } P_2 \in [\![x : B]\!]_\sigma\}^{\perp\perp} \\
[\![x : A \oplus B]\!]_\sigma &\triangleq \{P \mid \exists Q.\ P \approx x.\mathsf{inl}; Q \text{ and } Q \in [\![x : A]\!]_\sigma \text{ or} \\
&\qquad P \approx x.\mathsf{inr}; Q \text{ and } Q \in [\![x : B]\!]_\sigma\}^{\perp\perp} \\
[\![x : !A]\!]_\sigma &\triangleq \{P \mid \exists Q.\ P \approx !x(y); Q \text{ and } Q \in [\![y : A]\!]_\sigma\}^{\perp\perp} \\
[\![x : \exists X.A]\!]_\sigma &\triangleq \{P \mid \exists Q, S \in \mathcal{R}[- : B].\ P \approx \mathsf{sendty}\ x(B); Q \text{ and} \\
&\qquad Q \in [\![x : A]\!]_{\sigma[X \mapsto S]}\}^{\perp\perp} \\
[\![x : \mu X.\ A]\!]_\sigma &\triangleq (\bigcap\{S \in \mathcal{R}[- : \mu X.A] \mid \mathsf{unfold}_\mu\ x; [\![x : A]\!]_{\sigma[X \mapsto S]} \subseteq S\})^{\perp\perp} \\
[\![x : \wedge A]\!]_\sigma &\triangleq \{P \mid \exists Q.\ P \approx \mathsf{affine}\ x; Q \text{ and } Q \in [\![x : A]\!]_\sigma\}^{\perp\perp} \\
[\![x : \mathsf{S}_\bullet A]\!]_\sigma &\triangleq \{P \mid P \approx \mathsf{cell}\ x(y.[\![y : \wedge A]\!]_\sigma) \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : \mathsf{S}_\circ A]\!]_\sigma &\triangleq \{P \mid P \approx \mathsf{empty}\ x(y.[\![y : \wedge A]\!]_\sigma) \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : B]\!]_\sigma &\triangleq [\![x : \overline{B}]\!]_\sigma^\perp \ \text{(B negative type)}
\end{aligned}
$$

Fig. 9: Logical Predicate $[\![x : A]\!]_\sigma$.

4.1 Sharing a Linear Session

Our first example illustrates how objects subject to a linear usage protocol and satisfying an invariant may be shared among multiple concurrent clients by serialising linear usages using a mutex cell, alternating ownership from the cell to clients and back at the invariant state, a commonly used discipline to implement and reason about resource sharing (see, e.g., [38, 17, 9]).

We illustrate with a basic toggle switch with two states - On and Off - the resource invariant is the state Off, and two operations #turnOn and #turnOff that must be executed in strict linear sequence (Fig. 10). The toggle protocol, defined by type Off, offers the single option #turnOn, after which it evolves to On. Conversely, type On offers the single option #turnOff, after which it evolves to an affine Off. The toggle process at t is defined by two mutually corecursive processes $\mathsf{on}(t)$ and $\mathsf{off}(t)$, which define the expected behaviour, and comply with the types On and Off.

Process $\mathsf{main}()$ introduces a mutex cell c storing an affine toggle object at the invariant type $\wedge$Off. It then shares it with two concurrent clients, each acquires the toggle in the invariant type and uses the linear protocol independently. After their linear interaction, they put back the toggle, the type system ensures that this can only happen when the invariant (given by the cell type) holds. When they are done, both clients release their respective usages of c, which ultimately leads to the cell being deallocated and the (affine) toggle to be discarded.

We have also developed CLASS code for a generic (polymorphic) wrapper factory that, for any affine corecursive protocol, generates a wrapper to a general invariant-based sharing interface.

```
type corec Off = &{|#turnOn : On}
type corec On  = &{|#turnOff : ∧Off}

off(t) ⊢ t : Off
off(t) = case t {|#turnOn : on(t)}

on(t) ⊢ t : ∧On
on(t) = case t {|#turnOff :
                  affine t; off(t)}

client1(c) ⊢ c : S•Off
client1(c) = take c(t);
             #turnOn t; #turnOff t;
             put c(t); release c

client2(c) ⊢ c : S•Off
client2(c) = take c(t);
             #turnOn t; #turnOff t;
             #turnOn t; #turnOff t;
             put c(t); release c

main()  ⊢ ∅
main()  = cut {cell c(t.affine t; off(t))
            |c|
            share c {
               client1(c)
               ||
               client2(c)}}
```

Fig. 10: Sharing a Linear Toggle Switch

4.2 Linked Lists, Update In-Place

In this example, we show how inductive/coinductive types combine harmoniously with CLASS state modalities to type linked data structures with memory-efficient updates in-place. More specifically, we show how to code a linked list, parametric on the type A of its affine values, with an append in-place operation (Fig. 11). An object of type $\mathsf{SList}(A)$ is a (full) cell storing a $\mathsf{List}(A)$ object. An object of type $\mathsf{List}(A)$ is a session that either selects #Null (the list is empty), in which case it closes; or selects #Next, in which case it sends an affine session $\wedge A$ representing the head element and continues as the tail $\mathsf{SList}(A)$. Process $\mathsf{nil}(l)$ - defines an empty list at l - and process $\mathsf{cnext}(a, c, l)$ - constructs a nonempty list l with head a and tail c. For example, a list with elements a_1, a_2 stored at $c_1 : \mathsf{S}_{\bullet}\mathsf{List}(A)$ is represented

$$\mathsf{cut}\{\ \mathsf{cell}\ c_1(l_1.\mathsf{cnext}(a_1, c_2, l_1))\ |c_2|\ \mathsf{cell}\ c_2(l_2.\mathsf{cnext}(a_2, c_s, l_2))\ |c_s|\ \mathsf{cell}\ c_s(l_0.\mathsf{nil}(l_0))\}$$

Process $\mathsf{append}(c, l', c') \vdash c : \overline{\mathsf{SList}(A)}, l' : \overline{\mathsf{List}(A)}, c' : \mathsf{SList}(A)$ produces on c' the result of appending l (in place) to c. It takes the list l stored in c, and then performs case analysis on l. If l selects #Null, it simply replaces the previous null node of c by l' and forwards the updated cell c to the output c'. This corresponds to the recursion base case in which the list l is empty.

If l selects #Next, in which case l has at least one element, one receives at l the node element $a : \vee\overline{A}$, and corecursively call append l' to the tail $l : \overline{\mathsf{SList}(A)}$ and puts back in c element a and tail x "returned" by the call. Notice that x is exactly x (by forwarding), which was passed along linearly. Remarkably, the $\mathsf{append}(c, l', c')$ operation just defined may be safely applied concurrently to the same shared linked list, with the final result being the correct one (some serialisation of the appends), without deadlocks or livelocks. It is also interesting to see how the type system forbids a list to be appended to itself.

```
type rec SList(A) = S•List(A)
type rec List(A)  = ⊕{
                      |#Null : 1,
                      |#Next : ∧A ⊗ SList(A)}

nil(l) ⊢ l : ∧List(A)
nil(l) = affine l;
         #Null l;
         close l

cnext(a, c, l) ⊢ a: ∨ Ā, c: ‾SList(A)‾, l: ∧ List(A)
cnext(a, c, l) = affine l;
                 #Next l;
                 send l(a);
                 fwd l c
```

```
append(c, l′, c′) =
 take c(l);
 case l {
  |#Null :
    wait l;
    put c(l′);
    fwd c c′
  |#Next :
    recv l(a);
    cut {
      append(l, l′, x)
      |x|
      put c(y.cnext(a, x, y));
      fwd c c′
    }}
```

Fig. 11: A Linked List with an Append In-Place Operation.

We have also developed many other in-place operations on linked data structures, such as insertion sort, and other kinds of linked structures such as queues and binary search trees. In the next examples we discuss a shared queue ADT with a fine-grained locking discipline and O(1) enqueue and dequeue operations.

4.3 A Concurrent Shareable Buffered Channel

In this section, we illustrate increased degrees of sharing in a mutable data structure with various references pointing to different parts of it, how the CLASS type system may express interfaces that talk about different client views for using a stateful object, and the use of polymorphism to implement information hiding ensuring that client code will never break the representation invariants of stateful ADTs, particularly challenging when aliasing and sharing are involved.

More concretely, we consider a shareable buffered channel (Fig. 12), and provide a realistic and efficient implementation [52] based on a message queue represented by a linked list with update-in-place (cf. Section 4.2 above) and two independent pointers: one to the head of the list, used for receiving, and another to the tail, used for sending. The operations are executed in O(1) time. Moreover we provide a typing with two separate send and receive views, which may be used by an arbitrary number of concurrent clients. In particular, when the list is nonempty, both send and receive run in true concurrency (asynchronously), without blocking each other, thanks to fine-grained locking.

The buffered channel type $\mathsf{BChan}(M)$, where M is the type of messages, offers two views: $\mathsf{SendT}(M)$ and $\mathsf{RecvT}(M)$, interfaces for sender and receiver endpoint clients. These views are exposed with a par ($⅋$), since they share an underlying resourceful structure. In fact, they could not be exported using a tensor ($\otimes$); it is interesting to notice how the type system imposes these constraints, important

```
type BChan(M) = SendT(M) ⅋ RecvT(M)
type SendT(M) = ∃SV.!MenuS(M, SV) ⊗ SV
type RecvT(M) = ∃RV.!MenuR(M, RV) ⊗ RV

type MenuS(M, SV) = & {
  |#Send : SV ⊸ ∧M ⊸ SV,
  |#Share : SV ⊸ (SV ⅋ SV),
  |#Free : SV ⊸ 1 },

type MenuR(M, RV) = & {
  |#Recv : RV ⊸ (Maybe(∧M) ⊗ RV),
  |#Share : RV ⊸ (RV ⅋ RV),
  |#Free : RV ⊸ 1 }

Rep = SV = RV = S•SList(M)
```

```
msend(me) =
recv me(tailptr);
recv me(a);
take tailptr(c);
take c(l);
cut {
  cell c′(l)
  |c′|
  share c′ {
    put c(l′.cnext(a, c′, l′));
    release c′
    ||
    put tailptr(c′);
    send me(tailptr);
    close me}}
```

Fig. 12: A Concurrent Shareable Buffered Channel.

to ensure deadlock freedom. The representation type of both views is Rep = S$_\bullet$SList(M) (see Section 4.2), hidden behind the SV and RV existential types [28, 54]; sending clients use a cell storing a reference to the tail node of the queue; receiving clients use a cell storing a reference to the head node of the queue.

Clients use the buffer through references of abstract type SV and RV and replicated menus !MenuS(M, SV) and !MenuR(M, RV). Both menus export the options #Share and #Free to allow sharing and release of the views. To send, a client selects #Send, sends his handle (of opaque type SV), the message to send and receives the (linear) handle back. In this implementation, receive is non-blocking, so operation #Recv returns a Maybe($\wedge M$) value: the client receives either #Nothing (if the buffer is empty) or #Just followed by a message a, otherwise. In 4.6 we discuss the implementation, in CLASS, of (Hoare style) monitors with conditions, which would allow a blocking receive to be implemented.

Process msend(me) implements the #Send "method". It first receives the sending view handle (of concrete type Rep), which is a cell with the $tailptr$, and the message a to be sent. Then, a new cell c' with nil (l) is created, the current tail of the list c is updated with a new node storing a and pointing to c'. Finally, the $tailptr$ cell is updated to point to the new tail node c' of the linked list.

4.4 Dining Philosophers

A resource hierarchy solution for Dijkstra's dining philosophers problem [33] requires forks to be acquired in some defined order. To model such order in CLASS we "encode" it with an explicit (necessarily) acyclic structure, which informs the type system about the safety of a particular acquisition order. This allows us to define a correct concurrent implementation of the philosophers, that satisfies deadlock freedom by pure linear logic typing. More concretely, we

$$
\begin{array}{l}
\mathsf{putNull}(f, f') \vdash f : \mathsf{U}_{\circ}\overline{\mathsf{Node}}, f' : \mathsf{Fork} \\
\mathsf{putNull}(f, f') \triangleq \mathsf{put}\ f(n.\mathsf{null}(n)); \mathsf{fwd}\ f\ f' \\
\\
\mathsf{eat}(f, f') \vdash f : \overline{\mathsf{Fork}}, f' : \mathsf{Fork} \\
\mathsf{eat}(f, f') \triangleq \\
\quad \mathsf{take}\ f(n); \\
\quad \mathsf{case}\ n\ \{ \\
\quad\quad |\#\mathsf{Null} : \\
\quad\quad \mathsf{wait}\ n; \mathsf{putNull}(f, f') \\
\quad\quad |\#\mathsf{Next} : \\
\quad\quad \mathsf{take}\ n(m); \\
\quad\quad \mathsf{put}\ n(m); \mathsf{put}\ f(n'.\mathsf{next}(n, n')); \\
\quad\quad \mathsf{fwd}\ f\ f' \}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{eat2}(f, f') \vdash f : \overline{\mathsf{Fork}}, f' : \mathsf{Fork} \\
\mathsf{eat2}(f, f') \triangleq \\
\quad \mathsf{take}\ f(n); \\
\quad \mathsf{case}\ n\ \{ \\
\quad\quad |\#\mathsf{Null} : \\
\quad\quad \mathsf{wait}\ n; \mathsf{putNull}(f, f') \\
\quad\quad |\#\mathsf{Next} : \\
\quad\quad \mathsf{cut}\ \{ \\
\quad\quad\quad \mathsf{takeLast}(n, x) \\
\quad\quad\quad |x| \\
\quad\quad\quad \mathsf{recv}\ x(m); \mathsf{wait}\ x; \\
\quad\quad\quad \mathsf{put}\ f(n'.\mathsf{next}(m, n')); \\
\quad\quad\quad \mathsf{fwd}\ f\ f' \}
\end{array}
$$

Fig. 13: The Dining Philosophers.

organize the forks in a linked chain defined by the inductive types rec $\mathsf{Fork} = \mathsf{S}_{\bullet}\mathsf{Node}$ and rec $\mathsf{Node} = \oplus\{\#\mathsf{Null} : \mathbf{1}, \#\mathsf{Next} : \mathsf{Fork}\}$.

Any fork in the chain may be shared by an arbitrary number of philosophers, cocontraction ensures that philosophers cannot communicate between themselves via any other channel, all synchronisation must happen via the chained forks. If a philosopher successfully takes a fork f_i, he can then take any fork f_j, with $i < j$; crucially, he must follow the path dictated by the chain, hence cannot acquire forks f_j with $j < i$. In Fig. 13 we define the eat operation, which allows each philosopher P_i, with $0 \leq i < k-1$ to eat: it acquires two consecutive forks in the chain. And eat2, which is the specific eating operation for the symmetry breaker P_{k-1}: it acquires the first fork, and traverses the chain to acquire the last with $\mathsf{takeLast}(n, x) \vdash n : \overline{\mathsf{Fork}}, x : \mathsf{Fork} \otimes \mathbf{1}$.

4.5 A Barrier for N threads

We describe in Fig. 14 a CLASS implementation of a simple barrier, parametric on the number N of threads to synchronise. We find it interesting to model the "real" code shown in the Rust reference page for std::sync::Mutex [43]. The code uses if-then-else and primitive integers, supported by our implementation, but that could be defined as idioms of pure CLASS processes.

We represent a barrier by a mutex cell storing a pair consisting of an integer n, holding the number of threads that have not yet reached the barrier, and a stack s of waiting threads, each represented by a session of *affine* type $\wedge\bot$ (so they will be safely aborted if at least one thread fails to reach the barrier).

The type Barrier of the barrier is $\mathsf{S}_{\bullet}\mathsf{BState}$, where $\mathsf{BState} \triangleq \mathsf{Int} \otimes \wedge\mathsf{List}(\wedge\bot)$. Initially the barrier is initialised with $n = N$ threads and an empty stack, so that the invariant $n + depth(s) = N$ holds during execution. Each $\mathsf{thread}(c; i)$ acquires the barrier c and checks if it is the last thread to reach the barrier (if $n == 1$): in this case, it awakes all the waiting threads ($\mathsf{awakeAll}(w_s)$) and resets the barrier.

```
init(w_s) ⊢ w_s : ∧BState
init(w_s) ≜
   affine w_s; send w_s(N); affine w_s; nil(w_s)

awakeAll(w_s : ‾List(∧⊥)‾)
awakeAll(w_s) ≜
   case w_s {
      #Nil : wait w_s; 0
      #Cons :
      recv w_s(w);
      par {close w || awakeAll(w_s)}

spawnAll(c; i, n) ⊢ c : ‾Barrier‾; i : ‾Int‾, n : ‾Int‾
spawnAll(c; i, n) ≜
   if (n == 0) { release c}
   { share c {
         thread(c; i)
         ||
         spawnall(c; i + 1, n − 1)}}
```

```
thread(c; i) ⊢ c : ‾Barrier‾; i : ‾Int‾
thread(c; i) =
   println i + ": waiting.";
   take c(w_s); recv w_s(n);
   if (n == 1) {
      par {
         println i + ": finished.";
         awakeAll(w_s)
         ||
         put c(w'_s.init(w'_s));
         release c}}
   { cut {
         affine w; wait w;
         println i + ": finished."; 0
         |w| put c(w'_s.affine w'_s;
                     send w'_s(n − 1);
                     affine w'_s;
                     cons(w, w_s, w'_s));
         release c}}
```

Fig. 14: A Barrier for N Threads

Otherwise, it updates the barrier by decrementing n and pushing its continuation into the stack (the continuation for thread i just prints "finished"). The following process $\mathsf{main}() \vdash \emptyset$ creates a new barrier c and spawns N threads, each labelled by a unique id i: $\mathsf{main}() \triangleq \mathsf{cut}\ \{\ \mathsf{cell}\ c(w_s.\mathsf{init}(w_s))\ |c|\ \mathsf{spawnAll}(c; 0, N)\ \}$. Again, our type system statically ensures that the code does not deadlock or livelock.

4.6 A Hoare Style Monitor

A Hoare style monitor is a well-know powerful programming abstraction [38], allowing concurrent operations on shared data to be coordinated in a sound way, so that it always satisfy a correctness invariant. The key essential idea is that concurrent client threads use the monitor lock to access the protected state in mutual exclusion, but may also wait (via a *await* primitive) inside the monitor until the state satisfies specific (pre-)conditions, while transferring state ownership to other threads potentially responsible for establishing such conditions and announcing it (via a *notify* primitive).

We discuss a CLASS implementation of a monitor, sketching the main components and how they are typed (Fig. 15). We consider a counter with value n, with increment #Inc and decrement #Dec operations, and subject to the invariant $n \geq 0$. The type of the counter CounterI exposes two separate, coinductively defined, client interfaces DecI and IncI for decrementing and incrementing.

While the #Inc operation is synchronous, the #Dec operation is always called asynchronously by passing a continuation (of type ContDec). This allows decre-

```
type corec IncI ≜ &{|#Inc : IncI,  |#End : ⊥}
type corec DecI ≜
   ∨ & {|#Dec : ∨(ContDec ⊸ ⊥), #End : ⊥}
type corec ContDec ≜ ∨(DecI ⊗ 1)
type CounterI ≜ DecI ⅋ IncI

type rec Rep ≜ (!Int) ⊗ WaitQ
type rec WaitQ ≜ ∧ ⊕  {|#Null : 1,  |#Next : NodeQ}
type rec NodeQ ≜ S•(ContDecW ⊗ WaitQ)
type rec ContDecW ≜ ∧(∧Rep ⊸ ∧Rep ⊗ DecI ⊸ ⊥)
```

$$\mathsf{awaitNZ} \vdash m : \mathsf{U}_\circ\overline{\mathsf{Rep}},\; n : \overline{!\mathsf{Int}}, w : \overline{\mathsf{WaitQ}}, cc : \overline{\mathsf{ContDecW}}$$

$$\mathsf{notifyNZ} \vdash m : \mathsf{U}_\circ\overline{\mathsf{Rep}}, s : \overline{\mathsf{Rep}}, m' : \mathsf{S}_\bullet\mathsf{Rep}$$

$$\mathsf{incloop} \vdash iv : \overline{\mathsf{IncI}}, m : \mathsf{U}_\bullet\overline{\mathsf{Rep}}$$

```
awaitNZ(m, n, w, cc) ≜
  put m(w'.affine v;
    send w'(n);
    consWQ(cc, w, w'));
    release m

incloop(iv, m) ≜
  case iv {
  #Inc : take m(r);
         recv r(n);
     cut {
      send s(n + 1); fwd s r
      |s| notifyNZ(m, s, m')
      |m'| incloop(iv, m') }
  #End : wait iv; release m
} }
```

Fig. 15: Implementing a Counter Monitor with Await / Notify.

menters to wait inside the monitor for condition NZ ($n > 0$) when $n = 0$. The condition NZ is represented by a wait queue of type WaitQ. The representation type of the monitor (Rep) holds the counter value and the wait queue. Each node in the wait queue stores information, of type ContDecW, for the waiting thread. Every such ContDecW objects stores (1) the pending action on the internal monitor state (of type $\wedge$Rep $\multimap$ $\wedge$Rep), to be executed after await returns, and (2) a callback to the continuation provided by the external client in the asynchronous call (of type DecI $\multimap \bot$).

The awaitNZ(m, n, w, cc) process implements the monitor wait operation, used in the #Dec operation. It receives the (empty) cell usage m to the monitor state, the integer value n (where $n = 0$), a reference w to the wait queue, and the continuation cc, it pushes a new node in the queue and puts the monitor state back, unlocking the cell m, and releases m. The incloop(iv, m) process implements the counter IncI interface. The call to notifyNZ(m, s, m') after incrementing n will cause a waiting DecI thread to be awaken (if any), and continue by applying the pending action to the Rep state s in which $n > 0$ holds, before passing the updated state m' to the incloop recursive call. Affinity plays a key role, allowing all data structures, including waiting continuations to be safely discarded, at the end of any computation.

We have only shown here some code snippets, the complete code is available in our distribution. We have also implemented generic code to simplify the construction of monitors, eventually using several condition. It is interesting to see how our system types this non-trivial concurrent code, involving higher-order mutable state, rich sharing and ownership transfer patterns, ensuring deadlock, livelock freedom and memory safety of code akin to real system-level code.

5 Related Work

Many resource-aware logics and type systems to tame shared state and interference have been proposed [3, 53, 71, 41, 17, 56, 57, 23]. These systems adopt some form of linearity and/or affinity to resourceful programming [69, 29] and to model failures/exceptions [27, 55, 19, 35, 49]. In CLASS, linearity allows us to control state sharing, whereas affinity is useful for memory safety and to represent abortable computations. The hereditary session-discarding behaviour of affine sessions, modelled by rule [$\wedge\vee$d], is also present in other works, e.g. [6, 55, 19].

CLASS builds on top of the PaT correspondence with Linear Logic [21, 26, 74], the logical principles for the state modalities being inspired by DiLL [34]. Recent works [9, 10, 7, 47, 60, 62] also address the problem of sharing and nondeterminism in the setting of session-based PaT. In [62], reference cells may only store replicated sessions (of type $!A$), thus cannot refer to linear entities such as other cells or linear sessions, hence cannot represent many realistic programming idioms that CLASS does (see Section 4). Accommodating linear state in a pure PaT approach is thus addressed in this work with a novel, more fundamental approach. Furthermore, in [62], recursion is obtained from polymorphism [73], in the style of system-F encodings, and cannot represent inductive stateful structures with memory-efficient updates in-place, as we do in CLASS, using native inductive/coinductive types and recursion operators.

The take/put operations of CLASS relate with the acquire/release operations of the manifest sharing session-typed language SILL_S [9, 10]. Sharing in SILL_S is based on shift modalities to move from shared to linear mode and back, and contraction principles to alias shared sessions. In CLASS we explore DiLL modalities and cocontraction principles [34] to express sharing of linear state and put / take protocols of mutex memory cells of invariant type. As a consequence [10] ensures deadlock-freedom by relying on programmer provided partial orders on events [51, 32, 25], whereas in CLASS deadlock-freedom follows naturally as a deep consequence of linear logic cut and cocontraction, already expressed by the basic “lightweight” type system. The work [60] introduces the CSLL language, by extending linear logic with coexponentials that support a notion of shared state, with a quite different approach than ours. CSLL does not claim the ability to naturally express shared linked data structures with update in-place and fine-grained locking, as CLASS does. Nevertheless, it is natural to define in CLASS sessions exporting weakening, sharing and dereliction capabilities for linear behaviours, as in our shared buffer example. None of the models in [9, 10, 60] addresses livelock absence or memory safety, as CLASS does.

As far as we are aware, CLASS is a first proposal integrating shared state and recursion in a language based on PaT and Linear Logic, while guaranteeing strong normalisation. Least/greatest fixed points in Linear Logic were studied in [8], which inspired the development of recursion in [50, 67], our treatment of recursion draws inspiration on [67]. Several works exploit the technique of logical relations to establish strong normalisation for concurrent process calculi [1, 77, 63, 16, 58]. The work [16] proves strong normalisation for a language with higher-order store with a type and effect system that stratifies memory into regions so as to preclude circularities. Interestingly, in CLASS such stratification is implicitly guaranteed by the acyclicity inherent to Linear Logic. Linear logical relations were studied in [58, 20, 66, 68]. In this work we recast and extend the technique to Classical Linear Logic, exploring orthogonality [37, 8, 1], and demonstrate, using a specially devised technique of interference-sensitive reducibility, how logical relations scale to accommodate shared state.

6 Concluding Remarks

We have introduced CLASS, a session-based language founded on a propositions-as-types interpretation of Second-Order Classical Linear Logic, extended with recursion, affine types, first-class mutex cells and shared linear state. As a consequence of its logical foundations, we believe that CLASS is the first proposal of a language of its kind to provide the following three strong properties by static typing: well-typed CLASS programs enjoy progress, hence never deadlock, do not leak memory and always terminate. Besides the foundational relevance of our work, we also argued how CLASS can cleanly express realistic concurrent higher-order programming idioms, with many compelling examples: sharing of corecursive linear protocols, memory-efficient dynamic linked data structures with update in-place, shareable concurrent ADTs and resource synchronisation methods, such as barriers and monitors.

Any type system introduces conservative restrictions on its language, but we believe that CLASS offers an interesting balance between the strong properties it ensures by typing and its expressiveness. In fact, we find CLASS type system helpful to guide the development of safe concurrent idioms, with a fairly light type annotation burden. The linear logic discipline demands that no more than one bundle of linear resources may be shared by any two independent threads. Nevertheless, as our examples show, it is most often the case that concurrent programs may be conveniently structured in this way, so that the shared bundles of linear resources may be safely encapsulate and coordinated, in monitor-like structures, through clean informative interfaces.

The restriction to primitive recursion on inductive types may seem a limitation in some situations and perhaps one may even want sometimes to write non-terminating code, so it is reasonable to expect that any pragmatic language based on CLASS may provide some "unsafe recursion" mechanism. Nevertheless, being able to check that substantial parts of a codebase are not only deadlock but also terminating / livelock free by typing seems to be a desirable feature.

The feasibility of CLASS is corroborated by our implementation of a fully-fledged type checker and interpreter. The type checker provides substantial type inference and reconstruction abilities, and the interpreter includes efficient pragmatic basic datatypes. The system, together with an extensive CLASS library of code, including the examples in this paper, which were all validated by the implementation, will be submitted as a companion artifact for this paper.

As future work, we would like to investigate several possible refinements of the CLASS type discipline, namely, allowing finer-grained resource-access policies to be expressed, and exploring the integration of dependent and refinement types [65, 48], enhancing the logical expressiveness of the basic type system.

References

1. Abramsky, S.: Computational Interpretations of Linear Logic. Theoret. Comput. Sci. **111**(1–2), 3–57 (1993)
2. Abramsky, S., Gay, S.J., Nagarajan, R.: Interaction categories and the foundations of typed concurrent programming. In: NATO ASI DPD. pp. 35–113 (1996)
3. Ahmed, A., Fluet, M., Morrisett, G.: L^3: A linear language with locations. Fundam. Inf. **77**(4), 397–449 (Dec 2007)
4. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. J. Logic Comput. **2**(3), 197–347 (1992)
5. Andreoli, J.M.: Logic Programming with Focusing Proofs in Linear Logic. J. Log. Comput. **2**(3), 297–347 (1992)
6. Asperti, A., Roversi, L.: Intuitionistic light affine logic. ACM Transactions on Computational Logic (TOCL) **3**(1), 137–175 (2002)
7. Atkey, R., Lindley, S., Morris, J.G.: Conflation Confers Concurrency, pp. 32–55. Springer International Publishing, Cham (2016)
8. Baelde, D.: Least and greatest fixed points in linear logic. TOCL **13**(1) (Jan 2012)
9. Balzer, S., Pfenning, F.: Manifest sharing with session types. Proc. ACM Program. Lang. **1**(ICFP) (Aug 2017)
10. Balzer, S., Toninho, B., Pfenning, F.: Manifest deadlock-freedom for shared session types. In: Caires, L. (ed.) Programming Languages and Systems. pp. 611–639. Springer International Publishing, Cham (2019)
11. Barber, A.: Dual Intuitionistic Linear Logic. Tech. Rep. LFCS-96-347, Univ. of Edinburgh (1996)
12. Beffara, E.: A Concurrent Model for Linear Logic. ENTCS **155**, 147–168 (2006)
13. Beffara, E.: An algebraic process calculus. In: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science. p. 130–141. LICS '08, IEEE Computer Society, USA (2008)
14. Bellin, G., Scott, P.: On the π-calculus and linear logic. Theoret. Comput. Sci. **135**(1), 11–65 (1994)
15. Benton, P.N.: A mixed linear and non-linear logic: Proofs, terms and models. In: International Workshop on Computer Science Logic. pp. 121–135. Springer (1994)
16. Boudol, G.: Typing termination in a higher-order concurrent imperative language. Information and Computation **208**(6), 716–736 (2010)
17. Brookes, S., O'Hearn, P.W.: Concurrent Separation Logic. ACM SIGLOG News **3**(3), 47–65 (2016)
18. Caires, L., Pérez, J.A., Pfenning, F., Toninho, B.: Relational parametricity for polymorphic session types. Tech. Rep. CMU-CS-12-108, Carnegie Mellon Univ. (2012)
19. Caires, L., Pérez, J.A.: Linearity, control effects, and behavioral types. In: Proceedings of the 26th European Symposium on Programming Languages and Systems - Volume 10201. p. 229–259. Springer-Verlag, Berlin, Heidelberg (2017)
20. Caires, L., Pérez, J.A., Pfenning, F., Toninho, B.: Behavioral polymorphism and parametricity in session-based communication. In: Proceedings of the 22nd European Conference on Programming Languages and Systems. p. 330–349. ESOP'13, Springer-Verlag, Berlin, Heidelberg (2013)
21. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory. pp. 222–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

22. Caires, L., Pfenning, F., Toninho, B.: Towards concurrent type theory. In: Proceedings of the 8th ACM SIGPLAN Workshop on Types in Language Design and Implementation. p. 1–12. TLDI '12, Association for Computing Machinery, New York, NY, USA (2012)
23. Caires, L., Seco, J.a.C.: The type discipline of behavioral separation. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. p. 275–286. POPL '13, Association for Computing Machinery, New York, NY, USA (2013)
24. Caires, L., Vieira, H.T.: Conversation types. In: Castagna, G. (ed.) Programming Languages and Systems. pp. 285–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
25. Caires, L., Vieira, H.T.: Conversation types. Theor. Comput. Sci. **411**(51-52), 4399–4440 (2010)
26. Caires, L., Pfenning, F., Toninho, B.: Linear logic propositions as session types. Mathematical Structures in Computer Science **26**(3), 367–423 (2016)
27. Carbone, M., Honda, K., Yoshida, N.: Structured Interactional Exceptions in Session Types. In: CONCUR 2008. LNCS, vol. 5201, pp. 402–417. Springer (2008)
28. Cardelli, L., Wegner, P.: On understanding types, data abstraction, and polymorphism. ACM Computing Surveys (CSUR) **17**(4), 471–523 (1985)
29. Clarke, D.G., Potter, J.M., Noble, J.: Ownership types for flexible alias protection. In: Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. p. 48–64. OOPSLA '98, Association for Computing Machinery, New York, NY, USA (1998)
30. Dardha, O., Gay, S.J.: A new linear logic for deadlock-free session-typed processes. In: Baier, C., Dal Lago, U. (eds.) Foundations of Software Science and Computation Structures. pp. 91–109. Springer International Publishing, Cham (2018)
31. Demangeon, R., Hirschkoff, D., Sangiorgi, D.: Mobile processes and termination. In: Semantics and Algebraic Specification, pp. 250–273. Springer (2009)
32. Dezani-Ciancaglini, M., de'Liguoro, U., Yoshida, N.: On progress for structured communications. In: Barthe, G., Fournet, C. (eds.) Trustworthy Global Computing. pp. 257–275. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
33. Dijkstra, E.W.: Hierarchical ordering of sequential processes. In: The origin of concurrent programming, pp. 198–227. Springer (1971)
34. Ehrhard, T.: An introduction to differential linear logic: proof-nets, models and antiderivatives. Mathematical Structures in Computer Science **28**(7), 995–1060 (2018)
35. Fowler, S., Lindley, S., Morris, J.G., Decova, S.: Exceptional asynchronous session types: session types without tiers. Proceedings of the ACM on Programming Languages **3**(POPL), 1–29 (2019)
36. Girard, J.Y.: Linear logic. Theoret. Comput. Sci. **50**(1), 1–102 (1987)
37. Girard, J.Y.: Linear logic. Theoretical computer science **50**(1), 1–101 (1987)
38. Hoare, C.A.R.: Monitors: An operating system structuring concept. Commun. ACM **17**(10), 549–557 (1974)
39. Hoare, C.A.R.: Towards a theory of parallel programming. In: The origin of concurrent programming, pp. 231–244. Springer (1972)
40. Howard, W.A.: The formulae-as-types notion of construction. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 479–490. Academic Press (1980)
41. Jacobs, J., Balzer, S., Krebbers, R.: Connectivity graphs: a method for proving deadlock freedom based on separation logic. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022)

42. Jones, S.P., Gordon, A., Finne, S.: Concurrent Haskell. In: POPL. vol. 96, pp. 295–308. Citeseer (1996)
43. Klabnik, S., Nichols, C.: The Rust Programming Language (2021)
44. Kobayashi, N.: A type system for lock-free processes. Information and Computation **177**(2), 122–159 (2002)
45. Kobayashi, N.: A new type system for deadlock-free processes. In: International Conference on Concurrency Theory. pp. 233–247. Springer (2006)
46. Kobayashi, N., Sangiorgi, D.: A hybrid type system for lock-freedom of mobile processes. ACM Transactions on Programming Languages and Systems (TOPLAS) **32**(5), 1–49 (2008)
47. Kokke, W., Morris, J.G., Wadler, P.: Towards races in linear logic. In: Riis Nielson, H., Tuosto, E. (eds.) Coordination Models and Languages. pp. 37–53. Springer International Publishing, Cham (2019)
48. Krishnaswami, N.R., Pradic, P., Benton, N.: Integrating linear and dependent types. ACM SIGPLAN Notices **50**(1), 17–30 (2015)
49. Lagaillardie, N., Neykova, R., Yoshida, N.: Stay safe under panic: Affine rust programming with multiparty session types. arXiv preprint arXiv:2204.13464 (2022)
50. Lindley, S., Morris, J.G.: Talking bananas: structural recursion for session types. In: Garrigue, J., Keller, G., Sumii, E. (eds.) Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016. pp. 434–447. ACM (2016)
51. Lynch, N.A.: Fast allocation of nearby resources in a distributed system. In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing. p. 70–81. STOC '80, Association for Computing Machinery, New York, NY, USA (1980)
52. Marlow, S.: Parallel and concurrent programming in Haskell: Techniques for multicore and multithreaded programming. " O'Reilly Media, Inc." (2013)
53. Militão, F., Aldrich, J., Caires, L.: Aliasing control with view-based typestate. In: Proceedings of the 12th Workshop on Formal Techniques for Java-Like Programs. pp. 1–7 (2010)
54. Mitchell, J.C., Plotkin, G.D.: Abstract types have existential type. ACM Transactions on Programming Languages and Systems (TOPLAS) **10**(3), 470–502 (1988)
55. Mostrous, D., Vasconcelos, V.T.: Affine Sessions. In: Proc. of COORDINATION 2014. LNCS, vol. 8459, pp. 115–130. Springer (2014)
56. Nanevski, A., Morrisett, J.G., Birkedal, L.: Hoare type theory, polymorphism and separation. J. Funct. Program. **18**(5-6), 865–911 (2008)
57. O'Hearn, P.W., Reynolds, J.C.: From Algol to polymorphic linear lambda-calculus. J. ACM **47**(1), 167–223 (2000)
58. Pérez, J.A., Caires, L., Pfenning, F., Toninho, B.: Linear logical relations and observational equivalences for session-based concurrency. Information and Computation **239**, 254–302 (2014)
59. Pfenning, F.: Structural cut elimination. In: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science. p. 156. LICS '95, IEEE Computer Society, USA (1995)
60. Qian, Z., Kavvos, G.A., Birkedal, L.: Client-server sessions in linear logic **5**(ICFP) (Aug 2021)
61. Rocha, P., Caires, L.: A Propositions-as-Types System for Shared State. Tech. rep., NOVA Laboratory for Computer Science and Informatics (06 2021)
62. Rocha, P., Caires, L.: Propositions-as-types and shared state. Proceedings of the ACM on Programming Languages **5**(ICFP), 1–30 (2021)

63. Sangiorgi, D.: Termination of processes. Math. Struct. in Comp. Sci. **16**(1), 1–39 (2006)
64. Sangiorgi, D., Walker, D.: PI-Calculus: A Theory of Mobile Processes. Cambridge University Press, USA (2001)
65. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming. p. 161–172. PPDP '11, Association for Computing Machinery, New York, NY, USA (2011)
66. Toninho, B., Caires, L., Pfenning, F.: Corecursion and non-divergence in session-typed processes. In: Maffei, M., Tuosto, E. (eds.) TGC 2014. Lecture Notes in Computer Science, vol. 8902, pp. 159–175. Springer (2014)
67. Toninho, B., Caires, L., Pfenning, F.: Corecursion and non-divergence in session-typed processes. In: International Symposium on Trustworthy Global Computing. pp. 159–175. Springer (2014)
68. Toninho, B., Yoshida, N.: On polymorphic sessions and functions: A tale of two (fully abstract) encodings. ACM Trans. Program. Lang. Syst. **43**(2) (Jun 2021)
69. Tov, J.A., Pucella, R.: Practical Affine Types. In: POPL 2011. pp. 447–458 (2011)
70. Vieira, H.T., Vasconcelos, V.T.: Typing progress in communication-centred systems. In: International Conference on Coordination Languages and Models. pp. 236–250. Springer (2013)
71. Voinea, A.L., Dardha, O., Gay, S.J.: Resource sharing via capability-based multiparty session types. In: International Conference on Integrated Formal Methods. pp. 437–455. Springer (2019)
72. Wadler, P.: Linear types can change the world! In: Broy, M. (ed.) Proceedings of the IFIP Working Group 2.2, 2.3 Working Conference on Programming Concepts and Methods, 1990. p. 561. North-Holland (1990)
73. Wadler, P.: Recursive types for free (1990)
74. Wadler, P.: Propositions as sessions. In: Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming. p. 273–286. ICFP '12, Association for Computing Machinery, New York, NY, USA (2012)
75. Wadler, P.: Propositions as Sessions. Journal of Functional Programming **24**(2-3), 384–418 (2014)
76. Wadler, P.: Propositions as types. Communications of the ACM **58**(12), 75–84 (2015)
77. Yoshida, N., Berger, M., Honda, K.: Strong normalisation in the π-calculus. Information and Computation **191**(2), 145–202 (2004)

Appendix (Supplementary Material)

In Section A we present the type and process syntax, the type system and the operational semantics of CLASS. Then, we prove language safety by establishing type preservation in Section B and progress in Section C. We present the proof of strong normalisation in Section D.

A The Core Language **CLASS**

Definition A.1 (Types A). *Given a collection of type variables $X, Y, Z, \ldots$ we define types by*

$$
\begin{aligned}
A, B ::= \; & X \ \textit{(type variable)} \ \mid \\
& \mathbf{1} \ \textit{(one)} \mid \bot \ \textit{(bottom)} \mid \\
& A \otimes B \ \textit{(tensor)} \mid A \parr B \ \textit{(par)} \mid \\
& A \oplus B \ \textit{(plus)} \mid A \,\&\, B \ \textit{(with)} \\
& !A \ \textit{(bang)} \mid ?A \ \textit{(why not)} \mid \\
& \exists X.A \ \textit{(exists)} \mid \forall X.A \ \textit{(for all)} \\
& \mu X.\ A \ \textit{(mu)} \ \mid \nu X.\ A \ \textit{(nu)} \\
& \wedge A \ \textit{(affine)} \mid \ \vee A \ \textit{(coaffine)} \mid \\
& \mathsf{S}_{\bullet} A \ \textit{(full state)} \mid \mathsf{U}_{\bullet} A \ \textit{(full usage)} \mid \\
& \mathsf{S}_{\circ} A \ \textit{(empty state)} \mid \mathsf{U}_{\circ} A \ \textit{(empty usage)}
\end{aligned}
$$

Types are composed from type variables $(X, Y, Z, \ldots)$, units $(\mathbf{1}, \bot)$, multiplicatives $(\otimes, \parr)$, additives $(\oplus, \&)$, exponentials $(!, ?)$, second-order type quantifies $(\exists X., \forall X.)$, recursive types $(\mu X.\ , \nu X.\ )$, affine/co-affine modalities $(\wedge, \vee)$ and shared state modalities $(\mathsf{S}_{\bullet}, \mathsf{U}_{\bullet}, \mathsf{S}_{\circ}, \mathsf{U}_{\circ})$.

The expressions $\exists X.A$, $\forall X.A$, $\mu X.\ A$, $\nu X.\ A$ all bind the type variable X in A. All the other type variable occurrences are free. The set of free type variables of a type expression A is denoted by $\mathsf{fv}(A)$. We denote by $\{A/X\}B$ the type expression obtained by replacing the type variable X by A in B. We consider that the binary type connectives associate to the right, e.g. the type $A \otimes B \parr C$ should be parsed as $A \otimes (B \parr C)$. Furthermore, we consider that the unary type constructors have higher precedence that the binary connectives, e.g. the type $!A \otimes B$ should be parsed as $(!A) \otimes B$.

Definition A.2 (Duality on Types $\overline{A}$). *Duality $\overline{A}$ is the involution on types defined by*

$$
\begin{array}{lll}
\overline{\mathbf{1}} \ = \bot & \overline{A \otimes B} = \overline{A} \parr \overline{B} & \overline{A \oplus B} = \overline{A} \ \& \overline{B} \\
\overline{!A} \ = ?\overline{B} & \overline{\exists X.A} \ = \forall X.\overline{A} & \overline{\mu X.\ A} = \nu X.\ \{\overline{X}/X\}(\overline{A}) \\
\overline{\wedge A} = \vee \overline{A} & \overline{\mathsf{S}_{\bullet} A} \ = \mathsf{U}_{\bullet} \overline{A} & \overline{\mathsf{S}_{\circ} A} \ = \mathsf{U}_{\circ} \overline{A}
\end{array}
$$

$$
\begin{array}{rl}
P, Q ::= & \mathsf{0}\ \textit{(inaction)} \mid \\
& \mathsf{fwd}\ x\ y\ \textit{(forwarder)} \mid \\
& X(x, \vec{y})\ \textit{(variable)} \mid \\
& \mathcal{A}\ \textit{(action)} \mid \\
& P \mid\mid Q\ \textit{(mix)} \mid \\
& P\ |x : A|\ Q\ \textit{(cut)} \mid \\
& y.P\ |!x : A|\ Q\ \textit{(cut!)} \mid \\
& \mathsf{share}\ x\ \{P \mid\mid Q\}\ \textit{(share)} \mid \\
\mathcal{A}, \mathcal{B} ::= & \mathsf{close}\ x\ \textit{(close)} \mid \mathsf{wait}\ x; P\ \textit{(wait)} \mid \\
& x.\mathsf{inl}; P\ \textit{(choose left)} \mid x.\mathsf{inr}; P\ \textit{(choose right)} \\
& \mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ \textit{(offer)} \mid \\
& \mathsf{send}\ x(y.P); Q\ \textit{(send)} \mid \mathsf{recv}\ x(y); P\ \textit{(receive)} \mid \\
& !x(y); P\ \textit{(server)} \mid ?x; P\ \textit{(activation)} \mid \mathsf{call}\ x(y); P\ \textit{(call)} \mid \\
& \mathsf{sendty}\ x(A); P\ \textit{(type send)} \mid \mathsf{recvty}\ x(X); P\ \textit{(type receive)} \mid \\
& \mathsf{corec}\ X(z, \vec{w}); P\ [x, \vec{y}]\ \textit{(corec)} \\
& \mathsf{unfold}_{\mu}\ x; P\ \textit{(unfold } \mu\textit{)} \mid \mathsf{unfold}_{\nu}\ x; P\ \textit{(unfold } \nu\textit{)} \\
& \mathsf{affine}_{\vec{b}, \vec{c}}\ a; P\ \textit{(affine)} \mid \mathsf{discard}\ a\ \textit{(discard)} \mid \mathsf{use}\ a; P\ \textit{(use)} \mid \\
& \mathsf{cell}\ c(a.P)\ \textit{(full cell)} \mid \mathsf{release}\ c\ \textit{(free)} \mid \mathsf{take}\ c(a); P\ \textit{(take)} \\
& \mathsf{empty}\ c\ \textit{(empty)} \mid \mathsf{put}\ c(a.P); Q\ \textit{(put)}
\end{array}
$$

Fig. 16: Processes P of CLASS.

The lollipop type constructor is defined by $A \multimap B \triangleq \overline{A} \otimes B$. We write $\vec{x}$ to denote a finite (possibly empty) array of names.

Definition A.3 (Processes P). *The syntax of process terms for* CLASS *is defined in Fig. 16.*

The static part of the syntax comprises inaction, mix, cut, cut! and share; the dynamic part includes actions $\mathcal{A}, \mathcal{B}$, and forwarder. An action is typically a process $\alpha; P$, where α is an action-prefix and P is the continuation. An action is typically a process $\alpha; P$, where α is an action-prefix and P is the continuation. In these cases, the subject $s(\mathcal{A})$ of an action $\mathcal{A}$ is the leftmost name occurrence of $\mathcal{A}$. For example, the subject of the action $\mathsf{send}\ x(y.P); Q$ is x. The subject of $\mathsf{corec}\ X(z, \vec{w}); P\ [x, \vec{y}]$ is x.

The expression $P\ |x : A|\ Q$ binds the name x on processes P and Q. $y.P\ |!x : A|\ Q$ binds y in P and x in Q. Actions $\mathsf{send}\ x(y.P); Q$, $\mathsf{recv}\ x(y); P$, $!x(y); P$, $\mathsf{call}\ x(y); P$ bind y on P. Actions $\mathsf{cell}\ c(a.P)$, $\mathsf{take}\ c(a); P$, $\mathsf{put}\ c(a.P); Q$ bind name

a on process P. All other name occurrences are free. The set of free names of P is denoted by $\mathsf{fn}(P)$; if $\mathsf{fn}(P) = \emptyset$, we say P is closed. The expressions $\mathsf{recvty}\ x(X); P$ binds the type variable X on process P. All the other type variable occurrences are free. The set of free type variables of a process P is denoted by $\mathsf{fv}(P)$. Capture-avoiding substitution and α-conversion are defined as usual. We denote by $\{x/y\}P$ the process obtained by replacing the name y by x on P. Similarly, we denote by $\{A/X\}P$ the process term obtained by replacing type variable X by type expression A in process term P.

We write $\vec{A}$ to denote a finite (possibly empty) array of types. We write $\vec{x} : \vec{A}$, only if $\mathsf{length}(\vec{x}) = \mathsf{length}(\vec{A})$, to denote the typing assignment $\vec{x}[0] : \vec{A}[0], \ldots, \vec{x}[n-1] : \vec{A}[n-1]$, or $\emptyset$ in case $n = 0$. If $\vec{A}$ is an array of types with length n and $\mathcal{M}$ a type modality, then $\mathcal{M}\vec{A}$ is an array with length n and such that, for all $0 \leq i \leq n-1$, $(\mathcal{M}\ \vec{A})[i] = \mathcal{M}(\vec{A}[i])$. If $\vec{x}$ and $\vec{y}$ are arrays of names with the same length we let $\{\vec{x}/\vec{y}\}P$ denote the simultaneous substitution of each component $\vec{x}[i]$ by $\vec{y}[i]$ in process P.

A typing context is a finite partial assignment from names to types, which we denote by

$$\underbrace{x_1 : A_1, \ldots, x_n : A_n}_{\Delta}\ ;\ \underbrace{y_1 : B_1, \ldots, y_m : B_m}_{\Gamma}$$

Typing contexts are separated (with a semi-colon) into two parts: a linear part denoted by Δ and an unrestricted (or exponential) part, which absorbs weakening and contraction, and is denoted by Γ. The empty context is written $\emptyset$. We write Δ, Δ' (two comma-separated contexts) for the disjoint union of Δ and Δ'. The set of free type variables of a typing context is the union of the free type variables of the types in the image of the typing context. Typing judgments are of the form $P \vdash_\eta \Delta; \Gamma$ where P is a process, $\Delta; \Gamma$ is a typing context and η is a finite partial map

$$\eta = X_1(\vec{x_1}) \mapsto \Delta_1; \Gamma, \ldots, X_n(\vec{x_n}) \mapsto \Delta_n; \Gamma_n$$

where recursion variables are assigned to typing contexts.

If Γ is empty we write just $P \vdash_\eta \Delta$ instead of $P \vdash_\eta \Delta; \emptyset$. Similarly, if η is empty, we write $P \vdash \Delta; \Gamma$ instead of $P \vdash_\emptyset \Delta; \Gamma$. We define $\{y/x\}(\Delta; \Gamma)$ by cases: if $x \notin \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)$, then $\{y/x\}(\Delta; \Gamma) = \Delta; \Gamma$. If $\Delta = \Delta', x : A$, then $\{y/x\}(\Delta; \Gamma) = \Delta', y : A; \Gamma$. If $\Gamma = \Gamma', x : A$, then $\{y/x\}(\Delta; \Gamma) = \Delta; \Gamma', y : A$. We denote by $\{A/X\}(\Delta; \Gamma)$ the typing context obtained by replacing the free type variable X by A. Similarly, we extend simultaneous substitutions to typing contexts accordingly, written $\{\vec{x}/\vec{y}\}(\Delta; \Gamma)$.

Definition A.4. *The typing rules of* CLASS *are listed in Figs. 17, 18, 19, 20, 21. N.B.: In rule [T$\forall$], X does not occur free in $\Delta; \Gamma$.*

A process P is well-typed if $P \vdash_\eta \Delta; \Gamma$ for some typing contexts Δ and Γ and map η.

A process context $\mathcal{C}$ is a process expression containing a hole and it is defined in the usual way (see [64]). We write $-$ for the empty context and $\mathcal{C}[P]$ for the

$$\frac{}{\mathsf{0} \vdash_\eta \emptyset; \Gamma}\ [\mathrm{T0}] \qquad \frac{P \vdash_\eta \Delta'; \Gamma \quad Q \vdash_\eta \Delta; \Gamma}{P \mid\mid Q \ \vdash_\eta \Delta', \Delta; \Gamma}\ [\mathrm{Tmix}]$$

$$\frac{}{\mathsf{fwd}\ x\ y\ \vdash_\eta x : \overline{A}, y : A; \Gamma}\ [\mathrm{Tfwd}] \quad \frac{P \vdash_\eta \Delta', x : A; \Gamma \quad Q \vdash_\eta \Delta, x : \overline{A}; \Gamma}{P\ |x : A|\ Q\ \vdash_\eta \Delta', \Delta; \Gamma}\ [\mathrm{Tcut}]$$

$$\frac{}{\mathsf{close}\ x \vdash_\eta x : \mathbf{1}; \Gamma}\ [\mathrm{T}\mathbf{1}] \quad \frac{Q \vdash_\eta \Delta; \Gamma}{\mathsf{wait}\ x; Q\ \vdash_\eta \Delta, x : \bot; \Gamma}\ [\mathrm{T}\bot]$$

$$\frac{P_1 \vdash_\eta \Delta, x : A; \Gamma \quad P_2 \vdash_\eta \Delta, x : B; \Gamma}{\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}\ \vdash_\eta \Delta, x : A \,\&\, B; \Gamma}\ [\mathrm{T}\&]$$

$$\frac{Q_1 \vdash_\eta \Delta', x : A; \Gamma}{x.\mathsf{inl}; Q_1\ \vdash_\eta \Delta', x : A \oplus B; \Gamma}\ [\mathrm{T}\oplus_l] \quad \frac{Q_2 \vdash_\eta \Delta', x : B; \Gamma}{x.\mathsf{inr}; Q_2\ \vdash_\eta \Delta', x : A \oplus B; \Gamma}\ [\mathrm{T}\oplus_r]$$

$$\frac{P_1 \vdash_\eta \Delta_1, y : A; \Gamma \quad P_2 \vdash_\eta \Delta_2, x : B; \Gamma}{\mathsf{send}\ x(y.P_1); P_2\ \vdash_\eta \Delta_1, \Delta_2, x : A \otimes B; \Gamma}\ [\mathrm{T}\otimes]$$

$$\frac{Q \vdash_\eta \Delta, z : A, x : B; \Gamma}{\mathsf{recv}\ x(z); Q \vdash_\eta \Delta, x : A \parr B; \Gamma}\ [\mathrm{T}\parr]$$

$$\frac{P \vdash_\eta y : A; \Gamma}{!x(y); P\ \vdash_\eta x : !A; \Gamma}\ [\mathrm{T!}] \quad \frac{Q \vdash_\eta \Delta; \Gamma, x : A}{?x; Q\ \vdash_\eta \Delta, x : ?A; \Gamma}\ [\mathrm{T?}]$$

$$\frac{P \vdash_\eta y : A; \Gamma \quad Q \vdash_\eta \Delta; \Gamma, x : \overline{A}}{y.P\ |!x : A|\ Q\ \vdash_\eta \Delta; \Gamma}\ [\mathrm{Tcut!}] \quad \frac{Q \vdash_\eta \Delta, z : A; \Gamma, x : A}{\mathsf{call}\ x(z); Q\ \vdash_\eta \Delta; \Gamma, x : A}\ [\mathrm{Tcall}]$$

$$\frac{P \vdash_\eta \Delta, x : \{B/X\}A; \Gamma}{\mathsf{sendty}\ x(B); P\ \vdash_\eta \Delta, x : \exists X.A; \Gamma}\ [\mathrm{T}\exists] \quad \frac{Q \vdash_\eta \Delta, x : A; \Gamma}{\mathsf{recvty}\ x(X); Q \vdash_\eta \Delta, x : \forall X.A; \Gamma}\ [\mathrm{T}\forall]$$

Fig. 17: Typing Rules I: Second-Order CLL.

$$\frac{P \vdash_{\eta'} \Delta, z : A; \Gamma \quad \eta' = \eta, X(z, \vec{w}) \mapsto \Delta, z : Y; \Gamma}{\mathsf{corec}\ X(z, \vec{w}); P\ [x, \vec{y}] \vdash_\eta \{\vec{y}/\vec{w}\}\Delta, x : \nu Y.\ A; \{\vec{y}/\vec{w}\}\Gamma}\ [\mathrm{Tcorec}]$$

$$\frac{\eta = \eta', X(x, \vec{y})\ \mapsto \Delta, x : Y; \Gamma}{X(z, \vec{w}) \vdash_\eta \{\vec{w}/\vec{y}\}(\Delta, z : Y; \Gamma)}\ [\mathrm{Tvar}]$$

$$\frac{P \vdash_\eta \Delta, x : \{\nu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\nu\ x; P \vdash_\eta \Delta, x : \nu X.\ A; \Gamma}\ [\mathrm{T}\nu] \quad \frac{P \vdash_\eta \Delta, x : \{\mu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\mu\ x; P \vdash_\eta \Delta, x : \mu X.\ A; \Gamma}\ [\mathrm{T}\mu]$$

Fig. 18: Typing Rules II: Induction and Coinduction.

$$\frac{P \vdash_\eta \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}, a : A; \Gamma}{\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ \vdash_\eta \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}, a : \wedge A; \Gamma}\ [\text{Taffine}]$$

$$\frac{}{\mathsf{discard}\ a \vdash_\eta a : \vee A; \Gamma}\ [\text{Tdiscard}] \qquad \frac{Q \vdash_\eta \Delta, a : A; \Gamma}{\mathsf{use}\ a; Q \vdash_\eta \Delta, a : \vee A; \Gamma}\ [\text{Tuse}]$$

Fig. 19: Typing Rules III: Affinity.

$$\frac{P \vdash_\eta \Delta, a : \wedge A; \Gamma}{\mathsf{cell}\ c(a.P) \vdash_\eta \Delta, c : \mathsf{S}_\bullet A; \Gamma}\ [\text{Tcell}] \qquad \frac{}{\mathsf{release}\ c \vdash_\eta c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Trelease}]$$

$$\frac{}{\mathsf{empty}\ c \vdash_\eta c : \mathsf{S}_\circ A; \Gamma}\ [\text{Tempty}] \qquad \frac{Q \vdash_\eta \Delta, a : \vee A, c : \mathsf{U}_\circ A; \Gamma}{\mathsf{take}\ c(a); Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Ttake}]$$

$$\frac{Q_1 \vdash_\eta \Delta_1, a : \wedge\overline{A}; \Gamma \qquad Q_2 \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{put}\ c(a.Q_1); Q_2\ \vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\circ A; \Gamma}\ [\text{Tput}]$$

Fig. 20: Typing Rules IV: Reference Cells.

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\bullet A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\bullet A; \Gamma}\ [\text{Tsh}]$$

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\circ A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\circ A; \Gamma}\ [\text{TshL}]$$

$$\frac{P \vdash_\eta \Delta', c : \mathsf{U}_\bullet A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathsf{U}_\circ A; \Gamma}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathsf{U}_\circ A; \Gamma}\ [\text{TshR}]$$

Fig. 21: Typing Rules V: State Sharing.

$P \equiv P$	[refl]
$P \equiv Q \;\supset\; Q \equiv P$	[symm]
$P \equiv Q$ and $Q \equiv R \;\supset\; P \equiv R$	[trans]
$P \leq Q$ and $Q \leq R \;\supset\; P \leq R$	[trans2]
$P \equiv Q \;\supset\; \mathcal{C}[P] \equiv \mathcal{C}[Q]$	[cong]
$P \leq Q \;\supset\; \mathcal{C}[P] \leq \mathcal{C}[Q]$	[cong2]
$\mathsf{fwd}\ x\ y \;\equiv \mathsf{fwd}\ y\ x$	[fwd]
$P \mid x : A \mid Q \;\equiv Q \mid x : \overline{A} \mid P$	[C]
$P \parallel \mathsf{0} \;\equiv P$	[0M]
$P \parallel Q \equiv Q \parallel P$	[M]
$P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$	[MM]
$P \mid x : A \mid (Q \parallel R) \equiv (P \mid x : A \mid Q) \parallel R$	[CM]
$P \mid x : A \mid (Q \mid y : B \mid R) \equiv (P \mid x : A \mid Q) \mid y : B \mid R$	[CC]
$P \mid x : A \mid \mathsf{share}\ y\ \{Q \parallel R\} \equiv \mathsf{share}\ y\ \{P \mid x : A \mid Q \parallel R\}$	[CSh]
$P \mid z : A \mid (y.Q \mid !x : B \mid R) \equiv y.Q \mid !x : B \mid (P \mid z : A \mid R)$	[CC!]
$y.Q \mid !x : A \mid (P \parallel R) \equiv P(y.Q \mid !x : A \mid R) \parallel$	[C!M]
$y.P \mid !x : A \mid (w.Q \mid !z : B \mid R) \equiv w.Q \mid !z : B \mid (y.P \mid !x : A \mid R)$	[C!C!]
$\mathsf{share}\ x\ \{P \parallel (Q \parallel R)\} \equiv \mathsf{share}\ x\ \{P \parallel Q\} \parallel R$	[ShM]
$\mathsf{share}\ x\ \{P \parallel \mathsf{share}\ y\ \{Q \parallel R\}\} \equiv \mathsf{share}\ y\ \{\mathsf{share}\ x\ \{P \parallel Q\} \parallel R\}$	[ShSh]
$\mathsf{share}\ z\ \{P \parallel y.Q \mid !x : A \mid R\} \equiv y.Q \mid !x : A \mid \mathsf{share}\ z\ \{P \parallel R\}$	[ShC!]
$y.P \mid !x : A \mid (Q \parallel R) \equiv (y.P \mid !x : A \mid Q) \parallel (y.P \mid !x : A \mid R)$	[D-C!M]
$y.P \mid !x : A \mid (Q \mid z : B \mid R) \equiv (y.P \mid !x : A \mid Q) \mid z : B \mid (y.P \mid !x : A \mid R)$	[D-C!C]
$y.P \mid !x : A \mid (w.Q \mid !z : B \mid R)$ $\equiv w.(y.P \mid !x : A \mid Q) \mid !z : B \mid (y.P \mid !x : A \mid R)$	[D-C!C!]
$y.P \mid !x : A \mid \mathsf{share}\ z\ \{Q \parallel R\}$ $\equiv \mathsf{share}\ z\ \{(y.P \mid !x : A \mid Q) \parallel (y.P \mid !x : A \mid R)\}$	[D-C!Sh]
$\mathsf{share}\ x\ \{\mathsf{release}\ x \parallel P\} \leq P$	[ShRel]
$\mathsf{share}\ x\ \{\mathsf{put}\ x(y.P); Q \parallel R\} \leq \mathsf{put}\ x(y.P); \mathsf{share}\ x\ \{Q \parallel R\}$	[ShPut]
$\mathsf{share}\ x\ \{\mathsf{take}\ x(y_1); P \parallel \mathsf{take}\ x(y_2); P_2\}$ $\leq \mathsf{take}\ x(y_1); \mathsf{share}\ x\ \{P_1 \parallel \mathsf{take}\ x(y_2); P_2\}$	[ShTake]

Provisos: in [CM] and [ShM], $x \in \mathsf{fn}(Q)$; in [CC], [CSh] and [ShSh], $x, y \in \mathsf{fn}(Q)$; in [CC!], [C!M] and [ShC!], $x \notin \mathsf{fn}(P)$; in [C!C!], $x \notin \mathsf{fn}(Q)$ and $z \notin \mathsf{fn}(P)$.

Fig. 22: Structural congruence $P \equiv Q$ and precongruence $P \leq Q$.

$\mathsf{fwd}\ x\ y\ \lvert y:A\rvert\ P \to \{x/y\}P$	[fwd]
$\mathsf{close}\ x\ \lvert x:\mathbf{1}\rvert\ \mathsf{wait}\ x; P \to P$	[$\mathbf{1}\bot$]
$\mathsf{send}\ x(y.P); Q\ \lvert x:A\otimes B\rvert\ \mathsf{recv}\ x(z); R$ $\to Q\ \lvert x:B\rvert\ (P\ \lvert y:A\rvert\ \{y/z\}R)$	[$\otimes⅋$]
$\mathsf{case}\ x\ \{\lvert\mathsf{inl}: P,\ \lvert\mathsf{inr}: Q\}\ \lvert x:A\ \&\ B\rvert\ x.\mathsf{inl}; R \to P\ \lvert x:A\rvert\ R$	[$\&\oplus_l$]
$\mathsf{case}\ x\ \{\lvert\mathsf{inl}: P,\ \lvert\mathsf{inr}: Q\}\ \lvert x:A\ \&\ B\rvert\ x.\mathsf{inr}; R \to Q\ \lvert x:B\rvert\ R$	[$\&\oplus_r$]
$!x(y); P\ \lvert x:!A\rvert\ ?x; Q \to y.P\ \lvert !x:A\rvert\ Q$	[!?]
$y.P\ \lvert !x:A\rvert\ \mathsf{call}\ x(z); Q \to \{z/y\}P\ \lvert z:A\rvert\ (y.P\ \lvert !x:A\rvert\ Q)$	[call]
$\mathsf{sendty}\ x(A); P\ \lvert x:\exists X.\ B\rvert\ \mathsf{recvty}\ x(X); Q$ $\to P\ \lvert x:\{A/X\}B\rvert\ \{A/X\}Q$	[$\exists\forall$]
$\mathsf{unfold}_\mu\ x; P\ \lvert x:\mu X.\ A\rvert\ \mathsf{unfold}_\nu\ x; Q \to P\ \lvert x:\{\mu X.\ A/X\}A\rvert\ Q$	[$\mu\nu$]
$\mathsf{unfold}_\mu\ x; P\ \lvert x:\mu X.\ A\rvert\ \mathsf{corec}\ Y(z,\vec{w}); Q\ [x,\vec{y}]$ $\to P\ \lvert x:\{\mu X.\ A/X\}A\rvert\ \{x/z\}\{\vec{y}/\vec{w}\}\{\mathsf{corec}\ Y(z,\vec{w}); Q/Y\}Q$	[corec]
$\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ \lvert a:\wedge A\rvert\ \mathsf{discard}\ a \to \mathsf{discard}\ \vec{b}\ \vert\vert\ \mathsf{release}\ \vec{c}$	[$\wedge\vee$d]
$\mathsf{affine}_{\vec{b},\vec{c}}\ a; P\ \lvert a:\wedge A\rvert\ \mathsf{use}\ a; Q \to P\ \lvert a:A\rvert\ Q$	[$\wedge\vee$u]
$\mathsf{cell}\ c(a.P)\ \lvert c:\mathsf{S}_\bullet A\rvert\ \mathsf{release}\ c \to P\ \lvert a:\wedge A\rvert\ \mathsf{discard}\ a$	[$\mathsf{S}_\bullet\mathsf{U}_\bullet$r]
$\mathsf{cell}\ c(a.P)\ \lvert c:\mathsf{S}_\bullet A\rvert\ \mathsf{take}\ c(a'); Q$ $\to P\ \lvert a:\wedge A\rvert\ (\mathsf{empty}\ c\ \lvert c:\mathsf{S}_\circ A\rvert\ \{a/a'\}Q)$	[$\mathsf{S}_\bullet\mathsf{U}_\bullet$t]
$\mathsf{empty}\ c\ \lvert c:\mathsf{S}_\circ A\rvert\ \mathsf{put}\ c(a.P); Q \to \mathsf{cell}\ c(a.P)\ \lvert c:\mathsf{S}_\bullet A\rvert\ Q$	[$\mathsf{S}_\circ\mathsf{U}_\circ$]
$P \le P'$ and $P' \to Q'$ and $Q' \le Q\ \supset\ P \to Q$	[$\le$]
$P \to Q\ \supset\ \mathcal{C}[P]\ \to \mathcal{C}[Q]$	[cong]

Fig. 23: Reduction $P \to Q$.

process obtained by replacing the hole in $\mathcal{C}$ by P (notice that in $\mathcal{C}[P]$ the context $\mathcal{C}$ may bind free names of process P). Similarly, given two process contexts $\mathcal{C}_1, \mathcal{C}_2$, we write$\mathcal{C}_1[\mathcal{C}_2]$ for the context obtained by replacing the hole in $\mathcal{C}_1$ by $\mathcal{C}_2$. We define context composition by $\mathcal{C}_1 \circ \mathcal{C}_2 \triangleq \mathcal{C}_1[\mathcal{C}_2]$. A process P' is a subprocess of P if $P = \mathcal{C}[P']$, for some process context $\mathcal{C}$. We say that a relation $\mathcal{R}$ is a process congruence iff whenever $P\mathcal{R}Q$, then $\mathcal{C}[P]\mathcal{R}\mathcal{C}[Q]$.

Definition A.5 (Structural Congruence $P \equiv Q$). *Structural congruence $\equiv$ is the least congruence on processes closed under α-conversion and the $\equiv$-rules in Fig. 6. Structural precongruence $\leq$ is the least pre-congruence on processes including $\equiv$ and closed under α-conversion and the $\leq$-rules in Fig. 6.*

Before defining reduction, we introduce static contexts, which are defined by

$$\begin{array}{l}\mathcal{C} ::= - \mid \mathcal{C} \mid\mid P \mid P \mid\mid \mathcal{C} \mid \mathcal{C} \;|x|\; P \mid P \;|x|\; \mathcal{C} \mid y.P \;|!x|\; \mathcal{C} \;\mid \\ \quad\quad \mathsf{share}\ x\ \{\mathcal{C} \mid\mid P\} \mid \mathsf{share}\ x\ \{P \mid\mid \mathcal{C}\} \mid\end{array}$$

A static context is therefore a context where the hole is neither guarded by any action nor lies in the server body P of a cut! $y.P \;|!x|\; Q$.

We define $\mathsf{release}\ \vec{x}$ and $\mathsf{discard}\ \vec{x}$ by induction on $\vec{x}$:

$$\begin{array}{ll}\mathsf{release}\ [] \triangleq \mathsf{0} & \mathsf{release}\ (\vec{x} : y) \triangleq \mathsf{release}\ \vec{x} \mid\mid \mathsf{release}\ y \\ \mathsf{discard}\ [] \triangleq \mathsf{0} & \mathsf{discard}\ (\vec{x} : y) \triangleq \mathsf{discard}\ \vec{x} \mid\mid \mathsf{discard}\ y\end{array}$$

We need also to define substitution of a process variable by a corecursive process, which will be used when modelling the one-step unfold of a corecursive process definition. The base cases are defined by

$$\begin{array}{l}\{\mathsf{corec}\ X(z,\vec{w}); P/X\}X(x,\vec{y}) \triangleq \mathsf{corec}\ X(z,\vec{w}); P\ [x,\vec{y}] \\ \{\mathsf{corec}\ X(z,\vec{w}); P/X\}Y(x,\vec{y}) \triangleq Y(x,\vec{y}),\ Y \neq X\end{array}$$

and the substitution is propagated without surprises to the remaining cases.

Definition A.6 (Reduction $P \to Q$). *Reduction $\to$ is the least relation on processes that includes the rules in Fig.* **??**. *N.B.: In [cong], $\mathcal{C}$ is an arbitrary static context.*

We define $\Rightarrow$ as the transitive closure of $\to \cup \equiv$.

B Type Preservation

We prove type preservation for structural congruence $\equiv$ (Theorem B.1) and reduction $\to$ (Theorem B.2). But first we introduce some notation and prove some auxiliary lemmas.

B.1 Notation

Before presenting the complete proofs of type preservation for precongruence $\leq$ and reduction $\to$ we introduce some handy notations that make the presentation of the proofs more succinct.

State Flavours. We introduce two state flavours, namely e (empty) and f (full). If $\mathcal{X}$ is a flavour, then the metavariable type $\mathbf{S}_{\mathcal{X}}\ A$ denotes either the full cell modality $\mathsf{S}_{\bullet}A$, if $\mathcal{X} = f$, or either the empty cell modality $\mathsf{S}_{\circ}A$, if $\mathcal{X} = e$. Similarly, $\mathbf{U}_{\mathcal{X}}\ A$ denotes either $\mathsf{U}_{\bullet}A$, if $\mathcal{X} = f$, or $\mathsf{U}_{\circ}A$, if $\mathcal{X} = e$. Two flavours can be combined through a partial binary operation $\oplus$, defined by

$$f \oplus f \triangleq f \qquad f \oplus e \triangleq e \qquad e \oplus f \triangleq e$$

The operation $\oplus$ is commutative and associative, furthermore the value of an expression $\mathcal{X}_1 \oplus \ldots \oplus \mathcal{X}_n$ is either f, whenever all the $\mathcal{X}_i$ are f; or e, in case one and only one of the $\mathcal{X}_i$ is e.

With this notation at hand, we can succinctly group all the typing rules for sharing ([Tsh], [TshL], [TshR]) in a single typing rule schema

$$\frac{P \vdash_\eta \Delta', c : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma \quad Q \vdash_\eta \Delta, c : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma \quad \mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}}{\mathsf{share}\ c\ \{P \mid\mid Q\}\ \vdash_\eta \Delta', \Delta, c : \mathbf{U}_{\mathcal{X}}\ A; \Gamma}\ [\text{TshX}]$$

Type Inversion. Often, in the following proofs of type preservation and progress, we appeal to inversion principles for the typing relation. By inspecting the principal form, i.e. the outermost constructor, of a process P for which a typing judgement $P \vdash_\eta \Delta; \Gamma$ holds we can infer some particularities of the typing contexts Δ and Γ. This works because, by inspecting the principal form of the process P, we can infer which was the typing rule that was applied to the root of a derivation tree for $P \vdash_\eta \Delta; \Gamma$. For example, in a derivation for $P_1 \mid\mid P_2 \vdash_\eta \Delta; \Gamma$ the last rule has to be [Tmix], from which we conclude that there there are Δ_1, Δ_2 s.t. $\Delta = \Delta_1, \Delta_2$, $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$. To make the presentation succinct, in the following proofs, we refer to the corresponding inversion principle associated with a typing rule adding the superscript -1 to the typing rule name. So, for [Tmix], it would be [Tmix^{-1}].

B.2 Auxiliary Lemmas

We state some auxiliary lemmas which are used during the proofs of type preservation. The first lemma states that every subprocess of a well-typed process is well-typed. Furthermore, if we replace a subprocess Q of a process a well-typed process P by a subprocess Q' that types with the same typing context as Q, then the resulting substitution types with same typing context as P.

Lemma B.1. *Suppose $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$, for some process context $\mathcal{C}$. Then, there exists Δ', Γ' s.t.*

- *$P \vdash_\eta \Delta'; \Gamma$.*
- *For all $Q \vdash_\eta \Delta'; \Gamma'$, $\mathcal{C}[Q] \vdash_\eta \Delta'; \Gamma'$.*

Proof. If $\mathcal{C} = -$, then simply pick $\Delta' = \Delta$ and $\Gamma' = \Gamma$. The hypothesis for the cases in which $\mathcal{C} \neq -$ is established by induction on the typing derivation tree that establishes $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$.

We illustrate with some cases.

Case: [T**1**].
From $\mathcal{C}[P] = \mathsf{close}\ x$ we conclude that $\mathcal{C} = -$ and $P = \mathsf{close}\ x$. Holds vacuously.
Case: [Tmix].
We have $\mathcal{C}[P] \vdash_\eta \Delta_1, \Delta_2; \Gamma$, $\mathcal{C}[P] = P_1 \mid\mid P_2$, $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$.
Since $\mathcal{C}[P] = P_1 \mid\mid P_2$, either (i) $\mathcal{C} = \mathcal{C}' \mid\mid R$ or (ii) $\mathcal{C} = R \mid\mid \mathcal{C}'$.
We consider (i) holds. The analysis is similar for (ii).
By applying the i.h. $\mathcal{C}'[P] \vdash_\eta \Delta_1; \Gamma$ we infer the existence of Δ_1', Γ' s.t.
(a) $P \vdash_\eta \Delta_1'; \Gamma'$.
(b) $\mathcal{C}'[Q] \vdash_\eta \Delta_1; \Gamma$ for all $Q' \vdash_\eta \Delta_1'; \Gamma'$.
Let $Q' \vdash_\eta \Delta_1'; \Gamma'$. From (b), $\mathcal{C}'[Q] \vdash_\eta \Delta_1; \Gamma$.
Applying [Tmix] to $\mathcal{C}'[Q] \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$ yields

$$\mathcal{C}[Q] = \mathcal{C}'[Q] \mid\mid P_2 \vdash_\eta \Delta_1, \Delta_2; \Gamma$$

Some formulations of the session-based interpretations of Linear Logic (cf. Wadler's CP) have explicit typing rules for weakening and contraction of the exponential modalities $!, ?$. In CLASS weakening and contraction are absorbed by the unrestricted typing context: we can adjoin an arbitrary formula in Γ (Lemma B.2([Tweaken]) or substitute the use of one formula for another (Lemma B.2([Tcontract]). Furthermore, we have a kind of *reverse* weakening principle: if a formula is not being used in a derivation, we can remove it from the unrestricted context (Lemma B.2([Tstrength])), this property is often referred to as *strengthening*.

Lemma B.2. *The following principles hold:*

[Tweaken] *If* $P \vdash_\eta \Delta; \Gamma$ *and* $x \notin dom(\Delta) \cup dom(\Gamma)$*, then* $P \vdash_\eta \Delta; \Gamma, x : A$.
[Tcontract] *If* $P \vdash_\eta \Delta; \Gamma, x : A, y : A$*, then* $\{x/y\}P \vdash_{\{x/y\}\eta} \Delta; \Gamma, x : A$.
[Tstrength] *If* $P \vdash_\eta \Delta; \Gamma, x : A$ *and* $x \notin \mathsf{fn}(P)$*, then* $P \vdash_\eta \Delta; \Gamma$.

Proof. **[Tweaken]** By induction on derivation tree for $P \vdash_\eta \Delta; \Gamma$. We illustrate with some cases.
Case: [T$\mathsf{0}$].
We have the conclusion $\mathsf{0} \vdash_\eta \emptyset; \Gamma$. By applying [T$\mathsf{0}$] we obtain $\mathsf{0} \vdash_\eta \emptyset; \Gamma, x : A$.
Case: [Tmix].
We have the conclusion $P_1 \mid\mid P_2 \vdash_\eta \Delta_1, \Delta_2; \Gamma$ from the premisses $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$.
Applying i.h. to $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$ yields $P_1 \vdash_\eta \Delta_1; \Gamma, x : A$ and $P_2 \vdash_\eta \Delta_2; \Gamma, x : A$, respectively.
Applying [Tmix] to $P_1 \vdash_\eta \Delta_1; \Gamma, x : A$ and $P_2 \vdash_\eta \Delta_2; \Gamma, x : A$ yields $P_1 \mid\mid P_2 \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : A$.
[Tcontract] By induction on derivation tree for $P \vdash_\eta \Delta; \Gamma$. We illustrate with some cases.

Case: [Tmix].

We have the conclusion $P_1 \parallel P_2 \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : A, y : A$ from the premisses $P_1 \vdash_\eta \Delta_1; \Gamma, x : A, y : A$ and $P_2 \vdash_\eta \Delta_2; \Gamma, x : A, y : A$.
Applying i.h. to $P_1 \vdash_\eta \Delta_1; \Gamma, x : A, y : A$ and $P_2 \vdash_\eta \Delta_2; \Gamma, x : A, y : A$ yields $\{x/y\}P_1 \vdash_\eta \Delta_1; \Gamma, x : A$ and $\{x/y\}P_2 \vdash_\eta \Delta_2; \Gamma, x : A$, respectively.
Applying [Tmix] to $\{x/y\}P_1 \vdash_\eta \Delta_1; \Gamma, x : A$ and $\{x/y\}P_2 \vdash_\eta \Delta_2; \Gamma, x : A$ yields $\{x/y\}P_1 \parallel \{x/y\}P_2 \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : A$.
Finally, note that $\{x/y\}(P_1 \parallel P_2) = \{x/y\}P_1 \parallel \{x/y\}P_2$.

Case: [Tcall].

There are three cases to consider, depending on wether the subject z of the call action is x, y or neither x nor y.

Case: $z \neq x, y$.

We have the conclusion $\mathsf{call}\ z(w); Q \vdash_\eta \Delta; \Gamma$, from the premiss $Q \vdash_\eta \Delta, w : B; \Gamma, x : A, y : A, z : B$.
Applying i.h. to $Q \vdash_\eta \Delta, w : B; \Gamma, x : A, y : A, z : B$ yields $\{x/y\}Q \vdash_\eta \Delta, w : B; \Gamma, x : A, z : B$.
Applying [Tcall] to $\{x/y\}Q \vdash_\eta \Delta; \Gamma, x : A, , z : B$ yields

$$\mathsf{call}\ z(w); \{x/y\}Q \vdash_\eta \Delta; \Gamma, x : A, z : B$$

Finally, note that $\{x/y\}(\mathsf{call}\ z(w); Q) = \mathsf{call}\ z(w); \{x/y\}Q$.

Case: $z = x$.

We have the conclusion $\mathsf{call}\ x(w); Q \vdash_\eta \Delta; \Gamma$, from the premiss $Q \vdash_\eta \Delta, w : A; \Gamma, x : A, y : A$.
Applying i.h. to $Q \vdash_\eta \Delta, w : A; \Gamma, x : A, y : A$ yields $\{x/y\}Q \vdash_\eta \Delta, w : A; \Gamma, x : A$.
Applying [Tcall] to $\{x/y\}Q \vdash_\eta \Delta, w : A; \Gamma, x : A$ yields

$$\mathsf{call}\ x(w); \{x/y\}Q \vdash_\eta \Delta; \Gamma, x : A$$

Finally, note that $\{x/y\}(\mathsf{call}\ x(w); Q) = \mathsf{call}\ x(w); \{x/y\}Q$.

Case: $z = y$.

We have the conclusion $\mathsf{call}\ y(w); Q \vdash_\eta \Delta; \Gamma$, from the premiss $Q \vdash_\eta \Delta, w : A; \Gamma, x : A, y : A$.
Applying i.h. to $Q \vdash_\eta \Delta, w : A; \Gamma, x : A, y : A$ yields $\{x/y\}Q \vdash_\eta \Delta, w : A; \Gamma, x : A$.
Applying [Tcall] to $\{x/y\}Q \vdash_\eta \Delta, w : A; \Gamma, x : A$ (this time on x) yields $\mathsf{call}\ x(w); \{x/y\}Q \vdash_\eta \Delta; \Gamma, x : A$.
Finally, note that $\{x/y\}(\mathsf{call}\ y(w); Q) = \mathsf{call}\ x(w); \{x/y\}Q$.

[Tstrength] Similar to [Tweaken].

The proof of type preservation also depends on a couple of auxiliary properties, which we will introduce now. The first (Lemma B.3(1)) states that the domain of the linear typing context with which a process P types is always the same.

To introduce the second property (Lemma B.3(2)) we need the following definition. Let Δ, Δ' be two partial maps from names to types. We say that Δ *is contained in* Δ' *up to usage flavours* iff the following hold

(1) if $x : A \in \Delta$ and $A \neq \mathbf{U}_{\mathcal{X}}\ B$, then $x : A \in \Delta'$.
(2) if $x : \mathbf{U}_{\mathcal{X}}\ B$, then $x : \mathbf{U}_{\mathcal{Y}}\ B \in \Delta'$ for some usage flavour $\mathcal{Y}$.

We say that Δ *and* Δ' *are the same up to usage flavours* iff Δ is contained in Δ' up to to usage flavours and vice-versa: Δ' is contained in Δ up to usage flavours.

Lemma B.3. *The following properties hold*

(1) *If* $P \vdash_\eta \Delta; \Gamma$ *and* $P \vdash_\eta \Delta'; \Gamma'$ *then* $dom(\Delta) = dom(\Delta')$.
(2) *Suppose* $P \vdash_\eta \Delta; \Gamma$, $P \vdash_\eta \Delta'; \Gamma$ *and let* Δ, Δ' *be the same up to usage flavours. Then,* $\Delta = \Delta'$.

Proof. **(1)** By induction on P. We illustrate with some cases.
Case: $P = \mathsf{0}$.
Applying $[\mathrm{T}\mathsf{0}^{-1}]$ to $\mathsf{0} \vdash_\eta \Delta; \Gamma$ yields $\Delta = \emptyset$.
Applying $[\mathrm{T}\mathsf{0}^{-1}]$ to $\mathsf{0} \vdash_\eta \Delta;' \Gamma'$ yields $\Delta' = \emptyset$.
Then, $\mathrm{dom}(\Delta) = \emptyset = \mathrm{dom}(\Delta')$.
Case $P = \mathsf{fwd}\ x\ y$.
By applying $[\mathrm{Tfwd}^{-1}]$ to $\mathsf{fwd}\ x\ y\ \vdash_\eta \Delta; \Gamma$ we infer the existence of A s.t. $\Delta = x : \overline{A}, y : A$.
By applying $[\mathrm{Tfwd}^{-1}]$ to $\mathsf{fwd}\ x\ y\ \vdash_\eta \Delta'; \Gamma$ we infer the existence of B s.t. $\Delta' = x : \overline{B}, y : B$.
Then, $\mathrm{dom}(\Delta) = \{x, y\}\ = \mathrm{dom}(\Delta')$.
Case: $P = P_1 \mid\mid P_2$.
By applying $[\mathrm{Tmix}^{-1}]$ to $P_1 \mid\mid P_2\ \vdash_\eta \Delta; \Gamma$ we infer the existence of Δ_1, Δ_2 s.t. $\Delta = \Delta_1, \Delta_2$, $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_2 \vdash_\eta \Delta_2; \Gamma$.
By applying $[\mathrm{Tmix}^{-1}]$ to $P_1 \mid\mid P_2\ \vdash_\eta \Delta'; \Gamma'$ we infer the existence of Δ_1', Δ_2' s.t. $\Delta' = \Delta_1', \Delta_2'$, $P_1 \vdash_\eta \Delta_1'; \Gamma'$ and $P_2 \vdash_\eta \Delta_2'; \Gamma'$.
Applying i.h. to $P_1 \vdash_\eta \Delta_1; \Gamma$ and $P_1 \vdash_\eta \Delta_1'; \Gamma'$ yields $\mathrm{dom}(\Delta_1) = \mathrm{dom}(\Delta_1')$.
Applying i.h. to $P_2 \vdash_\eta \Delta_2; \Gamma'$ and $P_2 \vdash_\eta \Delta_2'; \Gamma'$ yields $\mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta_2')$.
Then, $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta_1) \cup \mathrm{dom}(\Delta_2) = \mathrm{dom}(\Delta_1') \cup \mathrm{dom}(\Delta_2') = \mathrm{dom}(\Delta')$.
Case: $P = ?x; P'$.
By applying $[\mathrm{T}?^{-1}]$ to $?x; P\ \vdash_\eta \Delta; \Gamma$ we infer the existence of Δ_0, A s.t $\Delta = \Delta_0, x :?A$ and $P \vdash_\eta \Delta_0; \Gamma, x : A$.
By applying $[\mathrm{T}?^{-1}]$ to $?x; P\ \vdash_\eta \Delta'; \Gamma'$ we infer the existence of Δ_0', B s.t $\Delta = \Delta_0', x :?B$ and $P \vdash_\eta \Delta_0'; \Gamma', x : B$.
Applying i.h. to $P \vdash_\eta \Delta_0; \Gamma, x : A$ and $P \vdash_\eta \Delta_0'; \Gamma', x : B$ yields $\mathrm{dom}(\Delta_0) = \mathrm{dom}(\Delta_0')$.
Then, $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta_0) \cup \{x\} = \mathrm{dom}(\Delta_0') \cup \{x\} = \mathrm{dom}(\Delta')$.

(2) By induction on P and case analysis on its principal form. We illustrate with some cases.
Case $P = \mathsf{fwd}\ x\ y$.
By $[\mathrm{Tfwd}^{-1}]$ and $\mathsf{fwd}\ x\ y \vdash \Delta; \Gamma$ we conclude that $\Delta = x : A, y : \overline{A}$ for some type A. By $[\mathrm{Tfwd}^{-1}]$ and $\mathsf{fwd}\ x\ y \vdash \Delta'; \Gamma$ we conclude that $\Delta' = x : B, y : \overline{B}$ for some type B.
Either A or $\overline{A}$ is not an usage modality. Suppose w.l.o.g. that $A \neq \mathbf{U}_{\mathcal{X}}\ B$.
Then $A = B$ and, as consequence, $\overline{A} = \overline{B}$.

Case $P = \mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$.

By [Tsh^{-1}] and $\mathsf{share}\ x\ \{P_1 \mid\mid P_2\} \vdash_\eta \Delta; \Gamma$ we conclude that exists $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$ s.t. (1) $P_1 \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$, (2) $P_2 \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$, (3) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ and (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$.

By [Tsh^{-1}] and $\mathsf{share}\ x\ \{P_1 \mid\mid P_2\} \vdash_\eta \Delta'; \Gamma$ we conclude that exists $\Delta'_1, \Delta'_2, A', \mathcal{X}'_1, \mathcal{X}'_2, \mathcal{X}'$ s.t. (1′) $P_1 \vdash_\eta \Delta'_1, x : \mathbf{U}_{\mathcal{X}'_1}\ A'; \Gamma$, (2′) $P_2 \vdash_\eta \Delta'_2, x : \mathbf{U}_{\mathcal{X}'_2}\ A'; \Gamma$, (3′) $\Delta' = \Delta'_1, \Delta'_2, x : \mathbf{U}'_{\mathcal{X}}\ A'$ and (4′) $\mathcal{X}'_1 \oplus \mathcal{X}'_2 = \mathcal{X}'$.

From (3), (3′) and since Δ, Δ' are the same up to usage flavours we obtain $A = A'$. Furthermore, since $\Delta_1 = \Delta \restriction (\mathsf{fn}(P_1) \setminus \{x\})$ and $\Delta'_1 = \Delta' \restriction (\mathsf{fn}(P_1) \setminus \{x\})$, we conclude that Δ_1, Δ'_1 are the same up to usage flavours. Similarly, we conclude that Δ_2, Δ'_2 are the same up to usage flavours.

Applying the i.h. to P_1, (1) and (1′) yields $\Delta_1 = \Delta'_1$ and $\mathcal{X}_1 = \mathcal{X}'_1$. Applying the i.h. to P_2, (2) and (2′) yields $\Delta_2 = \Delta'_2$ and $\mathcal{X}_2 = \mathcal{X}'_2$. Therefore, $\mathcal{X} = \mathcal{Y}$ and $\Delta = \Delta'$.

We conclude this section with a couple of auxiliary results that state how substitution (name by name, type variable by type, process variable by corecursive process definition) affect the typing relation.

Lemma B.4. *The following properties hold*

(1) *If* $P \vdash_\eta \Delta; \Gamma$ *and* $x \notin dom(\Delta) \cup dom(\Gamma)$, *then* $\{x/y\}P \vdash_\eta \{x/y\}(\Delta; \Gamma)$.
(2) *If* $P \vdash_\eta \Delta; \Gamma$, *then* $\{A/X\}P \vdash_{\{A/X\}\eta} \{A/X\}(\Delta; \Gamma)$.
(3) *Suppose* $\mathsf{corec}\ Y(z, \vec{w}); P\ [z, \vec{w}] \vdash_\eta \Delta, z : \nu X.\ A; \Gamma$, $\eta' = \eta'', Y(z, \vec{w}) \mapsto \Delta, z : X; \Gamma$ *for some* η'' *which extends* η, *and suppose* $Q \vdash_{\eta'} \Delta'; \Gamma'$. *Then,* $\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}Q\ \vdash_{\eta''} \{\nu X.\ A/X\}(\Delta'; \Gamma')$.

Proof. Properties (1) and (2) are by induction on a derivation for $P \vdash_\eta \Delta; \Gamma$.

Property (3) is by induction on a derivation for $Q \vdash_{\eta'} \Delta', z : B; \Gamma'$. The only way of introducing the type variable X in the context $\Delta'; \Gamma'$, with which Q types, is by appealing to rule [Tvar] on process variable Y. Consequently, if process variable Y does not occur free in Q, then the property holds trivially since $\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}Q = Q$ and $\{\nu X.\ A/X\}(\Delta'; \Gamma') = \Delta'; \Gamma'$. We illustrate the proof with some cases:

Case: [Tvar].

Then

$$\frac{\eta' = \eta'', Y(z, \vec{w}) \mapsto \Delta, z : X; \Gamma}{Y(x, \vec{y}) \vdash_{\eta'} \{\vec{y}/\vec{w}\}(\Delta, x : X; \Gamma)}\ [\mathrm{Tvar}]$$

where $Q = Y(x, \vec{y})$.
By def.

$$\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}Y(x, \vec{y}) = \ \mathsf{corec}\ Y(z, \vec{w}); P\ [x, \vec{y}]$$

Since, by hypothesis $\mathsf{corec}\ Y(z, \vec{w}); P\ [z, \vec{w}] \vdash_\eta \Delta, z : \nu X.\ A; \Gamma$ and η'' extends η, then $\mathsf{corec}\ Y(z, \vec{w}); P\ [z, \vec{w}] \vdash_{\eta''} \Delta, z : \nu X.\ A; \Gamma$.
By name renaming, $\mathsf{corec}\ Y(z, \vec{w}); P\ [x, \vec{y}] \vdash_{\eta''} \{\vec{y}/\vec{w}\}(\Delta, x : \nu X.\ A; \Gamma)$.

Case: [Tmix].
Then

$$\frac{Q_1 \vdash_{\eta'} \Delta_1'; \Gamma' \quad Q_2 \vdash_{\eta'} \Delta_2'; \Gamma'}{Q_1 \mathbin{||} Q_2 \ \vdash_{\eta'} \Delta_1', \Delta_2', ; \Gamma'} \text{[Tmix]}$$

where $Q = Q_1 \mathbin{||} Q_2$ and $\Delta' = \Delta_1', \Delta_2'$.
By def.

$$\begin{aligned}&\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}(Q_1 \mathbin{||} Q_2)\\ &= (\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}Q_1) \mathbin{||} (\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}Q_2)\end{aligned}$$

Applying i.h. to $Q_1 \vdash_{\eta'} \Delta_1'; \Gamma'$ yields (a) $\{\mathsf{corec}\, Y(z, \vec{w}); P/Y\}Q_1 \vdash_{\eta''} \{\nu X.\ A/X\}(\Delta_1'; \Gamma')$.
Applying i.h. to $Q_2 \vdash_{\eta'} \Delta_2'; \Gamma'$ yields (b) $\{\mathsf{corec}\, Y(z, \vec{w}); P/Y\}Q_2 \vdash_{\eta''} \{\nu X.\ A/X\}(\Delta_2'; \Gamma')$.
Applying [Tmix] to (a) and (b) yields

$$\{\mathsf{corec}\ Y(z, \vec{w}); P/Y\}(Q_1 \mathbin{||} Q_2) \vdash_{\eta''} \{\nu X.\ A/X\}(\Delta_1', \Delta_2'; \Gamma')$$

B.3 Type Preservation

We start with the proof of type preservation for precongruence (Theorem B.1)and then we move to the proof of type preservation for reduction (Theorem B.2).

Theorem B.1 (Type Preservation $\leq$). *If $P \vdash_\eta \Delta; \Gamma$ and $P \leq Q$, then $Q \vdash_\eta \Delta; \Gamma$.*

Proof. By induction on a derivation tree for $P \equiv Q$ and case analysis on the root rule. We consider an axiomatisation of $\equiv$ equivalent to Def. A.5 but in which we drop rule [symm] $P \equiv Q \ \supset \ Q \equiv P$ and assume that each commuting conversion holds from left-to-right and right-to-left.

Case: [refl], $P \equiv P$.
Follows immediately.
Case: [trans], $P \equiv Q$ and $Q \equiv R \ \supset \ P \equiv R$.
(1) $Q \vdash_\eta \Delta; \Gamma$ (i.h., $P \vdash_\eta \Delta; \Gamma$ and $P \equiv Q$)
(2) $R \vdash_\eta \Delta; \Gamma$ (i.h., (1) and $Q \equiv R$)

Similarly for [trans2].
Case: [cong], $P \equiv Q \supset \mathcal{C}[P] \equiv \mathcal{C}[Q]$.
(1) $P \vdash_\eta \Delta'; \Gamma'$, for some Δ', Γ' (Lemma B.1 and $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$)
(2) $Q \vdash_\eta \Delta'; \Gamma'$ (i.h., (1) and $P \equiv Q$)
(3) $\mathcal{C}[Q] \vdash_\eta \Delta; \Gamma$ (Lemma B.1, (1), (2) and $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$)

Similarly for [cong2].
Case: [fwd], $\mathsf{fwd}\ x\ y \ \equiv \mathsf{fwd}\ y\ x$.
(1) $\Delta = x : \overline{A}, y : A$ ([Tfwd^{-1}] and $\mathsf{fwd}\ x\ y \ \vdash_\eta \Delta; \Gamma$)
(2) $\mathsf{fwd}\ y\ x \ \vdash_\eta y : A, x : \overline{A}; \Gamma$ ([Tfwd])
(3) $\mathsf{fwd}\ y\ x \ \vdash_\eta \Delta; \Gamma$ ((1) and (2))

Case: [M], $P \parallel Q \equiv Q \parallel P$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1; \Gamma$ (3) $Q \vdash_\eta \Delta_2; \Gamma$, for some Δ_1, Δ_2
([Tmix^{-1}] and $P \parallel Q \vdash_\eta \Delta; \Gamma$)
(4) $Q \parallel P \vdash_\eta \Delta_2, \Delta_1; \Gamma$ ([Tmix], (3) and (2))
(5) $Q \parallel P \vdash_\eta \Delta; \Gamma$ ((1) and (4))

Case: [C], $P \; |x : A| \; Q \equiv Q \; |x : \overline{A}| \; P$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (3) $Q \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $P \; |x : A| \; Q \vdash_\eta \Delta; \Gamma$)
(4) $Q \; |x : \overline{A}| \; P \vdash_\eta \Delta_2, \Delta_1; \Gamma$ ([Tcut], (3) and (2))
(5) $Q \; |x : \overline{A}| \; P \vdash_\eta \Delta; \Gamma$ ((1) and (4))

Case: [Sh], $\mathsf{share}\ x \ \{P \parallel Q\} \equiv \mathsf{share}\ x \ \{Q \parallel P\}$.
(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}} \ A$ (2) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1} \ A; \Gamma$
(3) $Q \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2} \ A; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some Δ_1, Δ_2
([Tsh^{-1}] and $\mathsf{share}\ x \ \{P \parallel Q\} \vdash_\eta \Delta; \Gamma$)
(5) $\mathcal{X}_2 \oplus \mathcal{X}_1 = \mathcal{X}$ ($\oplus$ is commutative and (4))
(6) $\mathsf{share}\ x \ \{Q \parallel P\} \vdash_\eta \Delta_2, \Delta_1, x : \mathbf{U}_{\mathcal{X}} \ A; \Gamma$ ([Tsh], (3),(2) and (5))
(7) $\mathsf{share}\ x \ \{Q \parallel P\} \vdash_\eta \Delta; \Gamma$ ((1) and (6))

Case: [MM] left-to-right, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1; \Gamma$ (3) $Q \parallel R \vdash_\eta \Delta_2; \Gamma$, for some Δ_1, Δ_2
([Tmix^{-1}] and $P \parallel (Q \parallel R) \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_2 = \Delta_{21}, \Delta_{22}$ (5) $Q \vdash_\eta \Delta_{21}; \Gamma$ (6) $R \vdash_\eta \Delta_{22}; \Gamma$, for some Δ_{21}, Δ_{22}
([Tmix^{-1}] and (3))
(7) $P \parallel Q \vdash_\eta \Delta_1, \Delta_{21}; \Gamma$ ([Tmix], (2) and (5))
(8) $(P \parallel Q) \parallel R \vdash_\eta \Delta_1, \Delta_{21}, \Delta_{22}; \Gamma$ ([Tmix], (7) and (6))
(9) $\Delta_1, \Delta_{21}, \Delta_{22} = \Delta$ ((1) and (4))
(10) $(P \parallel Q) \parallel R \vdash_\eta \Delta; \Gamma$ ((8) and (9))

Case: [MM] right-to-left, $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$. Similar to case [MM] left-to-right.

Case: [CM] left-to-right, $P \; |x : A| \; (Q \parallel R) \equiv (P \; |x : A| \; Q) \parallel R, \ x \in \text{fn}(Q)$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (3) $Q \parallel R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $P \; |x : A| \; (Q \parallel R) \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_2, x : \overline{A} = \Delta_{21}, \Delta_{22}$ (5) $Q \vdash_\eta \Delta_{21}; \Gamma$ (6) $R \vdash_\eta \Delta_{22}; \Gamma$, for some Δ_{21}, Δ_{22}
([Tmix^{-1}] and (3))
(7) $\Delta_{21} = \Delta'_{21}, x : \overline{A}$, for some Δ'_{21} ((4), (5) and $x \in \mathsf{fn}(Q)$)
(8) $Q \vdash_\eta \Delta'_{21}, x : \overline{A}$ ((5) and (7))
(9) $P \; |x : A| \; Q \vdash_\eta \Delta_1, \Delta'_{21}; \Gamma$ ([Tcut], (2), (8))
(10) $(P \; |x : A| \; Q) \parallel R \vdash_\eta \Delta_1, \Delta'_{21}, \Delta_{22}; \Gamma$ ([Tmix], (9) and (6))
(11) $\Delta_1, \Delta'_{21}, \Delta_{22} = \Delta$ ((1), (4) and (7))
(12) $(P \; |x : A| \; Q) \parallel R \vdash_\eta \Delta; \Gamma$ ((10) and (11))

Case: [CM] right-to-left, $(P \; |x : A| \; Q) \parallel R \equiv P \; |x : A| \; (Q \parallel R), \ x \in \text{fn}(Q)$.

(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \mid x : A \mid Q \vdash_\eta \Delta_1; \Gamma$ (3) $R \vdash_\eta \Delta_2; \Gamma$, for some Δ_1, Δ_2 ([Tmix^{-1}] and $(P \mid x : A \mid Q) \parallel R \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1 = \Delta_{11}, \Delta_{12}$ (5) $P \vdash_\eta \Delta_{11}, x : A; \Gamma$ (6) $Q \vdash_\eta \Delta_{12}, x : \overline{A}; \Gamma$, for some Δ_{11}, Δ_{12} ([Tcut^{-1}] and (2))
(7) $Q \parallel R \vdash_\eta \Delta_{12}, x : \overline{A}, \Delta_2; \Gamma$ ([Tmix], (6) and (3))
(8) $P \mid x : A \mid (Q \parallel R) \vdash_\eta \Delta_{11}, \Delta_{12}, \Delta_2; \Gamma$ ([Tcut], (5) and (7))
(9) $\Delta_{11}, \Delta_{12}, \Delta_2 = \Delta$ ((4) and (1))
(10) $P \mid x : A \mid (Q \parallel R) \vdash_\eta \Delta; \Gamma$ ((8) and (9))

Case: [CC] left-to-right,

$$P \mid x : A \mid (Q \mid y : B \mid R) \equiv (P \mid x : A \mid Q) \mid y : B \mid R,\ x, y \in \text{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (3) $Q \mid y : B \mid R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $P \mid x : A \mid (Q \mid y : B \mid R) \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_2, x : \overline{A} = \Delta_{21}, \Delta_{22}$ (5) $Q \vdash_\eta \Delta_{21}, y : B; \Gamma$ (6) $R \vdash_\eta \Delta_{22}, y : \overline{B}; \Gamma$, for some Δ_{21}, Δ_{22} ([Tcut^{-1}] and (3))
(7) $\Delta_{21} = \Delta'_{21}, x : \overline{A}$, for some Δ'_{21} ((4), (5) and $x \in \mathsf{fn}(Q)$)
(8) $Q \vdash_\eta \Delta'_{21}, x : \overline{A}, y : B; \Gamma$ ((5) and (7))
(9) $P \mid x : A \mid Q \vdash_\eta \Delta_1, \Delta'_{21}, y : B; \Gamma$ ([Tcut], (2), (8))
(10) $(P \mid x : A \mid Q) \mid y : B \mid R \vdash_\eta \Delta_1, \Delta'_{21}, \Delta_{22}; \Gamma$ ([Tcut], (9) and (6))
(11) $\Delta_1, \Delta'_{21}, \Delta_{22} = \Delta$ ((1), (4) and (7))
(12) $(P \mid x : A \mid Q) \mid y : B \mid R \vdash_\eta \Delta; \Gamma$ ((10) and (11))

Case: [CC] right-to-left,

$$(P \mid x : A \mid Q) \mid y : B \mid R \equiv P \mid x : A \mid (Q \mid y : B \mid R),\ x, y \in \text{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \mid x : A \mid Q \vdash_\eta \Delta_1, y : B; \Gamma$ (3) $R \vdash_\eta \Delta_2, y : \overline{B}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $(P \mid x : A \mid Q) \mid y : B \mid R \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1, y : B = \Delta_{11}, \Delta_{12}$ (5) $P \vdash_\eta \Delta_{11}, x : A\Gamma$ (6) $Q \vdash_\eta \Delta_{12}, x : \overline{A}; \Gamma$, for some Δ_{11}, Δ_{12} ([Tcut^{-1}] and (2))
(7) $\Delta_{12} = \Delta'_{12}, y : B$, for some Δ'_{12} ((4), (6) and $y \in \mathsf{fn}(Q)$)
(8) $Q \vdash_\eta \Delta'_{12}, y : B, x : \overline{A}; \Gamma$ ((6) and (7))
(9) $Q \mid y : B \mid R \vdash_\eta \Delta'_{12}, x : \overline{A}, \Delta_2; \Gamma$ ([Tcut], (8) and (3))
(10) $P \mid x : A \mid (Q \mid y : B \mid R) \vdash_\eta \Delta_{11}, \Delta'_{12}, \Delta_2; \Gamma$ ([Tcut], (5) and (9))
(11) $\Delta_{11}, \Delta'_{12}, \Delta_2 = \Delta$ ((1), (4) and (7))
(12) $P \mid x : A \mid (Q \mid y : B \mid R) \vdash_\eta \Delta; \Gamma$ ((10) and (11))

Case: [CC!] left-to-right,

$$P \mid x : A \mid (y.Q \mid !z : B \mid R) \equiv y.Q \mid !z : B \mid (P \mid x : A \mid R),\ z \notin \mathsf{fn}(P)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (3) $y.Q \mid !z : B \mid R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $P \mid x : A \mid (y.Q \mid !z : B \mid R) \vdash_\eta \Delta; \Gamma$)
(4) $Q \vdash_\eta y : B; \Gamma$ (5) $R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma, z : \overline{B}$ ([Tcut!$^{-1}$] and (3))
(6) $P \vdash_\eta \Delta_1, x : A; \Gamma, z : \overline{B}$ (Lemma B.2([Tweaken]), (2) and $z \notin \mathsf{fn}(P)$)

(7) $P \mid x : A \mid R \vdash_\eta \Delta_1, \Delta_2; \Gamma, z : \overline{B}$ ([Tcut], (6) and (5))
(8) $y.Q \mid !z : B \mid (P \mid x : A \mid R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut!], (4) and (7))
(9) $y.Q \mid !z : B \mid (P \mid x : A \mid R) \vdash_\eta \Delta; \Gamma$ ((1) and (8))

Case: [CC!] right-to-left,

$$y.Q \mid !z : B \mid (P \mid x : A \mid R) \equiv P \mid x : A \mid (y.Q \mid !z : B \mid R),\ z \notin \mathsf{fn}(P)$$

(1) $Q \vdash_\eta y : B; \Gamma$ (2) $P \mid x : A \mid R \vdash_\eta \Delta; \Gamma, z : \overline{B}$
([Tcut!$^{-1}$] and $y.Q \mid !z : B \mid (P \mid x : A \mid R) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2$ (4) $P \vdash_\eta \Delta_1, x : A; \Gamma, z : \overline{B}$ (5) $R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma, z : \overline{B}$, for some Δ_1, Δ_2 ([Tcut!$^{-1}$] and (2))
(6) $y.Q \mid !z : B \mid R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$ ([Tcut!], (1) and (5))
(7) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (Lemma B.2([Tstrength], (4) and $z \notin \mathsf{fn}(P)$))
(8) $P \mid x : A \mid (y.Q \mid !z : B \mid R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (7) and (5))
(9) $P \mid x : A \mid (y.Q \mid !z : B \mid R) \vdash_\eta \Delta; \Gamma$ ((3) and (8))

Case: [C!M] left-to-right, $y.P \mid !x : A \mid (Q \mid\mid R) \equiv (y.P \mid !x : A \mid Q) \mid\mid R, x \notin \mathsf{fn}(R)$.
(1) $P \vdash_\eta y : A; \Gamma$ (2) $Q \mid\mid R \vdash_\eta \Delta; \Gamma, x : A$
([Tcut!$^{-1}$] and $y.P \mid !x : A \mid (Q \mid\mid R) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2$ (4) $Q \vdash_\eta \Delta_1; \Gamma, x : A$ (5) $R \vdash_\eta \Delta_2; \Gamma, x : A$, for some Δ_1, Δ_2 ([Tmix^{-1}] and (2))
(5) $y.P \mid !x : A \mid Q \vdash_\eta \Delta_1; \Gamma$ ([Tcut!], (1) and (4))
(6) $R \vdash_\eta \Delta_2; \Gamma$ (Lemma B.2([Tstrength]), (5) and $x \notin \mathsf{fn}(R)$)
(7) $(y.P \mid !x : A \mid Q) \mid\mid R \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tmix], (5) and (6))
(8) $(y.P \mid !x : A \mid Q) \mid\mid R \vdash_\eta \Delta; \Gamma$ ((3) and (7))

Case: [C!M] right-to-left, $(y.P \mid !x : A \mid Q) \mid\mid R \equiv y.P \mid !x : A \mid (Q \mid\mid R), x \notin \mathsf{fn}(R)$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $y.P \mid !x : A \mid Q \vdash_\eta \Delta_1; \Gamma$ (3) $R \vdash_\eta \Delta_2; \Gamma$
([Tmix^{-1}] and $(y.P \mid !x : A \mid Q) \mid\mid R \vdash_\eta \Delta; \Gamma$)
(4) $P \vdash_\eta y : A; \Gamma$ (5) $Q \vdash_\eta \Delta_1; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (2))
(6) $R \vdash_\eta \Delta_2; \Gamma, x : \overline{A}$ (Lemma B.2([Tweaken]) and (3))
(7) $Q \mid\mid R \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : \overline{A}$ ([Tmix], (5) and (6))
(8) $y.P \mid !x : A \mid (Q \mid\mid R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut!], (4) and (7))
(9) $y.P \mid !x : A \mid (Q \mid\mid R) \vdash_\eta \Delta; \Gamma$ ((1) and (8))

Case: [C!C!] left-to-right,

$$y.P \mid !x : A \mid (w.Q \mid !z : B \mid R) \equiv w.Q \mid !z : B \mid (y.P \mid !x : A \mid R), x \notin \mathsf{fn}(Q), z \notin \mathsf{fn}(P)$$

(1) $P \vdash_\eta y : A; \Gamma$ (2) $w.Q \mid !z : B \mid R \vdash_\eta \Delta; \Gamma, x : \overline{A}$
([Tcut!$^{-1}$] and $y.P \mid !x : A \mid (w.Q \mid !z : B \mid R) \vdash_\eta \Delta; \Gamma$)
(3) $Q \vdash_\eta w : B; \Gamma, x : \overline{A}$ (4) $R \vdash_\eta \Delta; \Gamma, x : \overline{A}, z : \overline{B}$ ([Tcut!$^{-1}$] and (2))
(5) $P \vdash_\eta y : A; \Gamma, z : \overline{B}$ (Lemma B.2([Tweaken]), (1) and $z \notin \mathsf{fn}(P)$)
(6) $y.P \mid !x : A \mid R \vdash_\eta \Delta; \Gamma, z : \overline{B}$ ([Tcut!], (5) and (4))
(7) $Q \vdash_\eta w : B; \Gamma$ (Lemma B.2([Tstrength]), (3) and $x \notin \mathsf{fn}(Q)$)
(8) $w.Q \mid !z : B \mid (y.P \mid !x : A \mid R) \vdash_\eta \Delta; \Gamma$ ([Tcut!], (7) and (6))

Case: [C!C!] right-to-left,

$$w.Q \;|!z : B|\; (y.P \;|!x : A|\; R) \equiv y.P \;|!x : A|\; (w.Q \;|!z : B|\; R), x \notin \mathsf{fn}(Q), z \notin \mathsf{fn}(P)$$

(1) $Q \vdash_\eta w : B; \Gamma$ (2) $y.P \;|!x : A|\; R \vdash_\eta \Delta; \Gamma, z : \overline{B}$
([Tcut!$^{-1}$] and $w.Q \;|!z : B|\; (y.P \;|!x : A|\; R) \vdash_\eta \Delta; \Gamma$)
(3) $P \vdash_\eta y : A; \Gamma, z : \overline{B}$ (4) $R \vdash_\eta \Delta; \Gamma, z : \overline{B}, x : \overline{A}$ ([Tcut!$^{-1}$] and (2))
(5) $Q \vdash_\eta w : B; \Gamma, x : \overline{A}$ (Lemma B.2([Tweaken]), (1) and $x \notin \mathsf{fn}(Q)$)
(6) $w.Q \;|!z : B|\; R \vdash_\eta \Delta; \Gamma, x : \overline{A}$ ([Tcut!], (5) and (4))
(7) $P \vdash_\eta y : A; \Gamma$ (Lemma B.2([Tstrength]), (3) and $z \notin \mathsf{fn}(P)$)
(8) $y.P \;|!x : A|\; (w.Q \;|!z : B|\; R) \vdash_\eta \Delta; \Gamma$ ([Tcut!], (7) and (6))

Case: [CSh] left-to-right,

$$P \;|x : A|\; \mathsf{share}\ y\ \{Q \;||\; R\} \equiv \mathsf{share}\ y\ \{P \;|x : A|\; Q \;||\; R\},\ x, y \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1, x : A; \Gamma$ (3) $\mathsf{share}\ y\ \{Q \;||\; R\} \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $P \;|x : A|\; (\mathsf{share}\ y\ \{Q \;||\; R\}) \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_2, x : \overline{A} = \Delta_{21}, \Delta_{22}, y : \mathbf{U}_{\mathcal{X}}\ B$ (5) $Q \vdash_\eta \Delta_{21}, y : \mathbf{U}_{\mathcal{X}_1}\ B; \Gamma$
(6) $R \vdash_\eta \Delta_{22}, y : \mathbf{U}_{\mathcal{X}_2}\ B; \Gamma$ (7) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$
, for some $\Delta_{21}, \Delta_{22}, B, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$ ([Tsh^{-1}] and (3))
(8) $\Delta_{21} = \Delta'_{21}, x : \overline{A}$, for some Δ'_{21} ((4), (5) and $x \in \mathsf{fn}(Q)$)
(9) $Q \vdash_\eta \Delta'_{21}, x : \overline{A}, y : \mathbf{U}_{\mathcal{X}_1}\ B; \Gamma$ ((5) and (8))
(10) $P \;|x : A|\; Q \vdash_\eta \Delta_1, \Delta'_{21}, y : \mathbf{U}_{\mathcal{X}_1}\ B; \Gamma$ ([Tcut], (2), (9))
(11) $\mathsf{share}\ y\ \{(P \;|x : A|\; Q) \;||\; R\} \vdash_\eta \Delta_1, \Delta'_{21}, \Delta_{22}, y : \mathbf{U}_{\mathcal{X}}\ B; \Gamma$
([Tsh], (10), (6) and (7))
(12) $\Delta_1, \Delta'_{21}, \Delta_{22}, y : \mathbf{U}_{\mathcal{X}}\ B = \Delta$ ((1), (4) and (8))
(13) $\mathsf{share}\ y\ \{(P \;|x : A|\; Q) \;||\; R\} \vdash_\eta \Delta; \Gamma$ ((11) and (12))

Case: [CSh] right-to-left,

$$\mathsf{share}\ y\ \{P \;|x : A|\; Q \;||\; R\} \equiv P \;|x : A|\; \mathsf{share}\ y\ \{Q \;||\; R\},\ x, y \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2, y : \mathbf{U}_{\mathcal{X}}\ B$ (2) $P \;|x : A|\; Q \vdash_\eta \Delta_1, y : \mathbf{U}_{\mathcal{X}_1}\ B; \Gamma$
(3) $R \vdash_\eta \Delta_2, y : \mathbf{U}_{\mathcal{X}_2}\ B; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, B, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ y\ \{P \;|x : A|\; Q \;||\; R\} \vdash_\eta \Delta; \Gamma$)
(5) $\Delta_1, y : \mathbf{U}_{\mathcal{X}_1}\ B = \Delta_{11}, \Delta_{12}$ (6) $P \vdash_\eta \Delta_{11}, x : A; \Gamma$ (7) $Q \vdash_\eta \Delta_{12}, x : \overline{A}; \Gamma$, for some Δ_{11}, Δ_{12} ([Tcut^{-1}] and (2))
(8) $\Delta_{12} = \Delta'_{12}, y : \mathbf{U}_{\mathcal{X}_1}\ B$, for some Δ'_{12} ((5), (7) and $y \in \mathsf{fn}(Q)$)
(9) $Q \vdash_\eta \Delta'_{12}, y : \mathbf{U}_{\mathcal{X}_1}\ B, x : \overline{A}; \Gamma$ ((7) and (8))
(10) $\mathsf{share}\ y\ \{Q \;||\; R\} \vdash_\eta \Delta'_{12}, x : \overline{A}, \Delta_2, y : \mathbf{U}_{\mathcal{X}}\ B; \Gamma$ ([Tsh], (9), (3) and (4))
(11) $P \;|x : A|\; (Q \;|y : B|\; R) \vdash_\eta \Delta_{11}, \Delta'_{12}, \Delta_2, y : \mathbf{U}_{\mathcal{X}}\ B; \Gamma$ ([Tcut], (6) and (10))
(12) $\Delta_{11}, \Delta'_{12}, \Delta_2, y : \mathbf{U}_{\mathcal{X}}\ B = \Delta$ ((1), (5) and (8))
(13) $P \;|x : A|\; (\mathsf{share}\ y\ \{Q \;||\; R\}) \vdash_\eta \Delta; \Gamma$ ((11) and (12))

Case: [ShM] left-to-right,

$$\mathsf{share}\ x\ \{P \;||\; (Q \;||\; R)\} \equiv \mathsf{share}\ x\ \{P \;||\; Q\} \;||\; R,\ x \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $Q \parallel R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{P \parallel (Q \parallel R)\} \vdash_\eta \Delta; \Gamma$)
(5) $\Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A = \Delta_{21}, \Delta_{22}$ (6) $Q \vdash_\eta \Delta_{21}; \Gamma$
(7) $R \vdash_\eta \Delta_{22}; \Gamma$, for some Δ_{21}, Δ_{22} ([Tmix^{-1}] and (3))
(8) $\Delta_{21} = \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}_2}\ A$, for some Δ'_{21} ((5), (6) and $x \in \mathsf{fn}(Q)$)
(9) $Q \vdash_\eta \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}_2}\ A$ ((6) and (8))
(10) $\mathsf{share}\ x\ \{P \parallel Q\} \vdash_\eta \Delta_1, \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma$ ([Tsh], (2), (9) and (4))
(11) $(\mathsf{share}\ x\ \{P \parallel Q\}) \parallel R \vdash_\eta \Delta_1, \Delta'_{21}, \Delta_{22}, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma$ ([Tmix], (10) and (7))
(12) $\Delta_1, \Delta'_{21}, \Delta_{22}, x : \mathbf{U}_{\mathcal{X}}\ A = \Delta$ ((1), (5) and (8))
(13) $(\mathsf{share}\ x\ \{P \parallel Q\}) \parallel R \vdash_\eta \Delta; \Gamma$ ((11) and (12))

Case: [ShM] right-to-left,

$$\mathsf{share}\ x\ \{P \parallel Q\} \parallel R \equiv \mathsf{share}\ x\ \{P \parallel (Q \parallel R)\},\ x \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{share}\ x\ \{P \parallel Q\} \vdash_\eta \Delta_1; \Gamma$ (3) $R \vdash_\eta \Delta_2; \Gamma$, for some Δ_1, Δ_2
([Tmix^{-1}] and $(\mathsf{share}\ x\ \{P \parallel Q\}) \parallel R \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1 = \Delta_{11}, \Delta_{12}, x : \mathbf{U}_{\mathcal{X}}\ A$ (5) $P \vdash_\eta \Delta_{11}, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(6) $Q \vdash_\eta \Delta_{12}, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (7) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_{11}, \Delta_{12}, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and (2))
(8) $Q \parallel R \vdash_\eta \Delta_{12}, x : \mathbf{U}_{\mathcal{X}_2}\ A, \Delta_2; \Gamma$ ([Tmix], (6) and (3))
(9) $\mathsf{share}\ x\ \{P \parallel (Q \parallel R)\} \vdash_\eta \Delta_{11}, \Delta_{12}, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma$ ([Tsh], (5) and (8))
(10) $\Delta_{11}, \Delta_{12}, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A = \Delta$ ((4) and (1))
(11) $\mathsf{share}\ x\ \{P \parallel (Q \parallel R)\} \vdash_\eta \Delta; \Gamma$ ((9) and (10))

Case: [ShC!] left-to-right,

$$\mathsf{share}\ x\ \{P \parallel (y.Q\ |!z : B|\ R)\} \equiv y.Q\ |!z : B|\ (\mathsf{share}\ x\ \{P \parallel R\}),\ z \notin \mathrm{fn}(P)$$

(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $y.Q\ |!z : B|\ R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$
(4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{P \parallel (y.Q\ |!z : B|\ R)\} \vdash_\eta \Delta; \Gamma$)
(5) $Q \vdash_\eta y : B; \Gamma$ (6) $R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma, z : \overline{B}$ ([Tcut!$^{-1}$] and (3))
(7) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma, z : \overline{B}$ (Lemma B.2([Tweaken]), (2) and $z \notin \mathsf{fn}(P)$)
(8) $\mathsf{share}\ x\ \{P \parallel R\} \vdash_\eta \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma, z : \overline{B}$ ([Tsh], (7), (6) and (4))
(9) $y.Q\ |!z : B|\ (\mathsf{share}\ x\ \{P \parallel R\}) \vdash_\eta \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma$ ([Tcut!], (5) and (8))
(10) $y.Q\ |!z : B|\ (\mathsf{share}\ x\ \{P \parallel R\}) \vdash_\eta \Delta; \Gamma$ ((1) and (9))

Case: [ShC!] right-to-left,

$$y.Q\ |!z : B|\ (\mathsf{share}\ x\ \{P \parallel R\}) \equiv \mathsf{share}\ x\ \{P \parallel (y.Q\ |!z : B|\ R)\},\ z \notin \mathrm{fn}(P)$$

(1) $Q \vdash_\eta y : B; \Gamma$ (2) $\mathsf{share}\ x\ \{P \parallel R\} \vdash_\eta \Delta; \Gamma, z : \overline{B}$
([Tcut!$^{-1}$] and $y.Q\ |!z : B|\ (\mathsf{share}\ x\ \{P \parallel R\}) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (4) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma, z : \overline{B}$

(5) $R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma, z : \overline{B}$ (6) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and (2))
(7) $y.Q\ |!z : B|\ R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ ([Tcut!], (1) and (5))
(8) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$ (Lemma B.2([Tstrength]), (4) and $z \notin \mathsf{fn}(P)$)
(9) $\mathsf{share}\ x\ \{P \mid\mid (y.Q\ |!z : B|\ R)\} \vdash_\eta \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A; \Gamma$
([Tsh], (8), (7) and (6))
(10) $\mathsf{share}\ x\ \{P \mid\mid (y.Q\ |!z : B|\ R)\} \vdash_\eta \Delta; \Gamma$ ((3) and (9))

Case: [ShSh] left-to-right,

$$\mathsf{share}\, x\, \{P \mid\mid (\mathsf{share}\, y\, \{Q \mid\mid R\})\} \equiv \mathsf{share}\, y\, \{(\mathsf{share}\, x\, \{P \mid\mid Q\}) \mid\mid R\},\ x, y \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $P \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $\mathsf{share}\ y\ \{Q \mid\mid R\} \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$
(4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{P \mid\mid (\mathsf{share}\ y\ \{Q \mid\mid R\})\} \vdash_\eta \Delta; \Gamma$)
(5) $\Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A = \Delta_{21}, \Delta_{22}, y : \mathbf{U}_{\mathcal{Y}}\ B$ (6) $Q \vdash_\eta \Delta_{21}, y : \mathbf{U}_{\mathcal{Y}_1}\ B; \Gamma$
(7) $R \vdash_\eta \Delta_{22}, y : \mathbf{U}_{\mathcal{Y}_2}\ B; \Gamma$ (8) $\mathcal{Y}_1 \oplus \mathcal{Y}_2 = \mathcal{Y}$, for some $\Delta_{21}, \Delta_{22}, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}$
([Tsh^{-1}] and (3))
(9) $\Delta_{21} = \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}_2}\ A$, for some Δ'_{21} ((5), (6) and $x \in \mathsf{fn}(Q)$)
(10) $Q \vdash_\eta \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}_2}\ A, y : \mathbf{U}_{\mathcal{Y}_1}\ B; \Gamma$ ((6) and (9))
(11) $\mathsf{share}\, x\, \{P \mid\mid Q\} \vdash_\eta \Delta_1, \Delta'_{21}, x : \mathbf{U}_{\mathcal{X}}\ A, y : \mathbf{U}_{\mathcal{Y}_1}\ B; \Gamma$ ([Tsh], (2), (10) an (4))
(12) $\mathsf{share}\, y\, \{(\mathsf{share}\, x\, \{P \mid\mid Q\}) \mid\mid R\} \vdash_\eta \Delta_1, \Delta'_{21}, \Delta_{22}, x : \mathbf{U}_{\mathcal{X}}\ A, y : \mathbf{U}_{\mathcal{Y}}\ B; \Gamma$
([Tsh], (11), (7) and (8))
(13) $\Delta_1, \Delta'_{21}, \Delta_{22}, x : \mathbf{U}_{\mathcal{X}}\ A, y : \mathbf{U}_{\mathcal{Y}}\ B\ = \Delta$ ((1), (5) and (9))
(14) $\mathsf{share}\ y\ \{(\mathsf{share}\ x\ \{P \mid\mid Q\}) \mid\mid R\} \vdash_\eta \Delta; \Gamma$ ((11) and (12))

Case: [ShSh] right-to-left,

$$\mathsf{share}\, y\, \{(\mathsf{share}\, x\, \{P \mid\mid Q\}) \mid\mid R\} \equiv \mathsf{share}\, x\, \{P \mid\mid (\mathsf{share}\, y\, \{Q \mid\mid R\})\},\ x, y \in \mathrm{fn}(Q)$$

(1) $\Delta = \Delta_1, \Delta_2, y : \mathbf{U}_{\mathcal{Y}}\ B$ (2) $\mathsf{share}\ x\ \{P \mid\mid Q\} \vdash_\eta \Delta_1, y : \mathbf{U}_{\mathcal{Y}_1}\ B; \Gamma$
(3) $R \vdash_\eta \Delta_2, y : \mathbf{U}_{\mathcal{Y}_2}\ B; \Gamma$ (4) $\mathcal{Y}_1 \oplus \mathcal{Y}_2 = \mathcal{Y}$, for some $\Delta_1, \Delta_2, B, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}$
([Tsh^{-1}] and $\mathsf{share}\ y\ \{(\mathsf{share}\ x\ \{P \mid\mid Q\}) \mid\mid R\} \vdash_\eta \Delta; \Gamma$)
(5) $\Delta_1, y : \mathbf{U}_{\mathcal{Y}_1}\ B = \Delta_{11}, \Delta_{12}, x : \mathbf{U}_{\mathcal{X}}\ A$ (6) $P \vdash_\eta \Delta_{11}, x : \mathbf{U}_{\mathcal{X}_1}\ A\Gamma$
(7) $Q \vdash_\eta \Delta_{12}, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (8) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_{11}, \Delta_{12}, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and (2))
(9) $\Delta_{12} = \Delta'_{12}, y : \mathbf{U}_{\mathcal{Y}_1}\ B$, for some Δ'_{12} ((5), (7) and $y \in \mathsf{fn}(Q)$)
(10) $Q \vdash_\eta \Delta'_{12}, y : \mathbf{U}_{\mathcal{Y}_1}\ B, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ ((7) and (9))
(11) $\mathsf{share}\, y\, \{Q \mid\mid R\} \vdash_\eta \Delta'_{12}, x : \mathbf{U}_{\mathcal{X}_2}\ A, y : \mathbf{U}_{\mathcal{Y}}\ B, \Delta_2; \Gamma$ ([Tsh], (10), (3) and (4))
(12) $\mathsf{share}\ x\ \{P \mid\mid (\mathsf{share}\ y\ \{Q \mid\mid R\})\} \vdash_\eta \Delta_{11}, \Delta'_{12}, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A, y : \mathbf{U}_{\mathcal{Y}}\ B; \Gamma$
([Tsh], (6) and (11))
(13) $\Delta_{11}, \Delta'_{12}, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A, y : \mathbf{U}_{\mathcal{Y}}\ B = \Delta$ ((1), (5) and (9))
(14) $\mathsf{share}\ x\ \{P \mid\mid (\mathsf{share}\ y\ \{Q \mid\mid R\})\} \vdash_\eta \Delta; \Gamma$ ((12) and (13))

Case: [D-C!M] left-to-right,

$$y.P\ |!x : A|\ (Q \mid\mid R) \equiv (y.P\ |!x : A|\ Q) \mid\mid (y.P\ |!x : A|\ R)$$

(1) $P \vdash_\eta y : A; \Gamma$ (2) $Q \parallel R \vdash_\eta \Delta; \Gamma, x : \overline{A}$
([Tcut!$^{-1}$] and $y.P \,|!x : A|\, (Q \parallel R) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2$ (4) $Q \vdash_\eta \Delta_1; \Gamma, x : \overline{A}$ (5) $R \vdash_\eta \Delta_2; \Gamma, x : \overline{A}$, for some Δ_1, Δ_2
([Tmix^{-1}] and (2))
(6) $y.P \,|!x : A|\, Q \vdash_\eta \Delta_1; \Gamma$ ([Tcut!], (1) and (4))
(7) $y.P \,|!x : A|\, R \vdash_\eta \Delta_2; \Gamma$ ([Tcut!], (1) and (5))
(8) $(y.P \,|!x : A|\, Q) \parallel (y.P \,|!x : A|\, R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tmix], (6) and (7))
(9) $(y.P \,|!x : A|\, Q) \parallel (y.P \,|!x : A|\, R) \vdash_\eta \Delta; \Gamma$ ((3) and (8))

Case: [D-C!M] right-to-left,

$$(y.P \,|!x : A|\, Q) \parallel (y.P \,|!x : A|\, R) \equiv y.P \,|!x : A|\, (Q \parallel R)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $y.P \,|!x : A|\, Q \vdash_\eta \Delta_1; \Gamma$
(3) $y.P \,|!x : A|\, R \vdash_\eta \Delta_2; \Gamma$, for some Δ_1, Δ_2
([Tmix^{-1}] and $(y.P \,|!x : A|\, Q) \parallel (y.P \,|!x : A|\, R) \vdash_\eta \Delta; \Gamma$)
(4) $P \vdash_\eta y : A; \Gamma$ (5) $Q \vdash_\eta \Delta_1; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (2))
(6) $R \vdash_\eta \Delta_2; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (3))
(7) $Q \parallel R \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : \overline{A}$ ([Tmix], (5) and (6))
(8) $y.P \,|!x : A|\, (Q \parallel R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut!], (4) and (7))
(9) $y.P \,|!x : A|\, (Q \parallel R) \vdash_\eta \Delta; \Gamma$ ((1) and (8))

Case: [D-C!C] left-to-right,

$$y.P \,|!x : A|\, (Q \,|z : B|\, R) \equiv (y.P \,|!x : A|\, Q) \,|z : B|\, (y.P \,|!x : A|\, R)$$

(1) $P \vdash_\eta y : A; \Gamma$ (2) $Q \,|z : B|\, R \vdash_\eta \Delta; \Gamma, x : \overline{A}$
([Tcut!$^{-1}$ and $y.P \,|!x : A|\, (Q \,|z : B|\, R) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2$ (4) $Q \vdash_\eta \Delta_1, z : B; \Gamma, x : \overline{A}$ (5)
$R \vdash_\eta \Delta_2, z : \overline{B}; \Gamma, x : \overline{A}$, for some Δ_1, Δ_2
([Tcut!$^{-1}$] and (2))
(6) $y.P \,|!x : A|\, Q \vdash_\eta \Delta_1, z : B; \Gamma$ ([Tcut!], (1) and (4))
(7) $y.P \,|!x : A|\, R \vdash_\eta \Delta_2, z : \overline{B}; \Gamma$ ([Tcut!], (1) and (5))
(8) $(y.P \,|!x : A|\, Q) \,|z : B|\, (y.P \,|!x : A|\, R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (6) and (7))
(9) $(y.P \,|!x : A|\, Q) \,|z : B|\, (y.P \,|!x : A|\, R) \vdash_\eta \Delta; \Gamma$ ((3) and (8))

Case: [D-C!C] right-to-left,

$$(y.P \,|!x : A|\, Q) \,|z : B|\, (y.P \,|!x : A|\, R) \equiv y.P \,|!x : A|\, (Q \,|z : B|\, R)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $y.P \,|!x : A|\, Q \vdash_\eta \Delta_1, z : B; \Gamma$
(3) $y.P \,|!x : A|\, R \vdash_\eta \Delta_2, z : \overline{B}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $(y.P \,|!x : A|\, Q) \,|z : B|\, (y.P \,|!x : A|\, R) \vdash_\eta \Delta; \Gamma$)
(4) $P \vdash_\eta y : A; \Gamma$ (5) $Q \vdash_\eta \Delta_1, z : B; \Gamma, x : \overline{A}$ ([Tcut^{-1}] and (2))
(6) $R \vdash_\eta \Delta_2, z : \overline{B}; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (3))
(7) $Q \,|z : B|\, R \vdash_\eta \Delta_1, \Delta_2; \Gamma, x : \overline{A}$ ([Tcut], (5) and (6))
(8) $y.P \,|!x : A|\, (Q \,|z|\, R) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut!], (4) and (7))
(9) $y.P \,|!x : A|\, (Q \,|z|\, R) \vdash_\eta \Delta; \Gamma$ ((1) and (8))

Case: [D-C!C!] left-to-right,

$$y.P \; |!x : A| \; (w.Q \; |!z : B| \; R) \equiv w.(y.P \; |!x : A| \; Q) \; |!z : B| \; (y.P \; |!x : A| \; R)$$

(1) $P \vdash_\eta y : A; \Gamma$ (2) $w.Q \; |!z : B| \; R \; \vdash_\eta \Delta; \Gamma, x : \overline{A}$
([Tcut!$^{-1}$] and $y.P \; |!x : A| \; (w.Q \; |!z : B| \; R) \vdash_\eta \Delta; \Gamma$)
(3) $Q \vdash_\eta w : B; \Gamma, x : \overline{A}$ (4) $R \vdash_\eta \Delta; \Gamma, x : \overline{A}, z : \overline{B}$ ([Tcut!$^{-1}$] and (2))
(5) $y.P \; |!x : A| \; Q \vdash_\eta w : B; \Gamma$ ([Tcut!] (1) and (3))
(6) $y.P \; |!x : A| \; R \vdash_\eta \Delta; \Gamma, z : \overline{B}$ ([Tcut!], (1) and (4))
(7) $w.(y.P \; |!x : A| \; Q) \; |!z : B| \; (y.P \; |!x : A| \; R) \vdash_\eta \Delta; \Gamma$ ([Tcut!], (5) and (6))

Case: [D-C!C!] right-to-left,

$$w.(y.P \; |!x : A| \; Q) \; |!z : B| \; (y.P \; |!x : A| \; R) \equiv y.P \; |!x : A| \; (w.Q \; |!z : B| \; R)$$

(1) $y.P \; |!x : A| \; Q \vdash_\eta w : B; \Gamma$ (2) $y.P \; |!x : A| \; R \vdash_\eta \Delta; \Gamma, z : \overline{B}$
([Tcut!$^{-1}$] and $w.(y.P \; |!x : A| \; Q) \; |!z : B| \; (y.P \; |!x : A| \; R) \vdash_\eta \Delta; \Gamma$)
(3) $P \vdash_\eta y : A; \Gamma$ (4) $Q \vdash_\eta w : B; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (1))
(5) $R \vdash_\eta \Delta; \Gamma, z : \overline{B}, x : \overline{A}$ ([Tcut!$^{-1}$] and (2))
(6) $w.Q \; |!z : B| \; R \; \vdash_\eta \Delta; \Gamma, x : \overline{A}$ ([Tcut!], (4) and (5))
(7) $y.P \; |!x : A| \; (w.Q \; |!z| \; R) \vdash_\eta \Delta; \Gamma$ ([Tcut!], (3) and (6))

Case: [D-C!Sh] left-to-right,

$$y.P \; |!x : A| \; \mathsf{share} \; z \; \{Q \parallel R\} \equiv \mathsf{share} \; z \; \{(y.P \; |!x : A| \; Q) \parallel (y.P \; |!x : A| \; R)\}$$

(1) $P \vdash_\eta y : A; \Gamma$ (2) $\mathsf{share} \; z \; \{Q \parallel R\} \; \vdash_\eta \Delta; \Gamma, x : \overline{A}$
([Tcut!$^{-1}$] and $y.P \; |!x : A| \; (\mathsf{share} \; z \; \{Q \parallel R\}) \vdash_\eta \Delta; \Gamma$)
(3) $\Delta = \Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{Y}} \; B$ (4) $Q \vdash_\eta \Delta_1, z : \mathbf{U}_{\mathcal{Y}_1} \; B; \Gamma, x : \overline{A}$
(5) $R \vdash_\eta \Delta_2, z : \mathbf{U}_{\mathcal{Y}_2} \; B; \Gamma, x : \overline{A}$ (6) $\mathcal{Y}_1 \oplus \mathcal{Y}_2 = \mathcal{Y}$, for some $\Delta_1, \Delta_2, B, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}$
([Tsh^{-1}] and (2))
(7) $y.P \; |!x : A| \; Q \vdash_\eta \Delta_1, z : \mathbf{U}_{\mathcal{Y}_1} \; B; \Gamma$ ([Tcut!], (1) and (4))
(8) $y.P \; |!x : A| \; R \vdash_\eta \Delta_2, z : \mathbf{U}_{\mathcal{Y}_2} \; B; \Gamma$ ([Tcut!], (1) and (5))
(9) $\mathsf{share} \; z \; \{(y.P \; |!x : A| \; Q) \parallel (y.P \; |!x : A| \; R)\} \vdash_\eta \Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{Y}} \; B; \Gamma$
([Tsh], (7), (8) and (6))
(10) $\mathsf{share} \; z \; \{(y.P \; |!x : A| \; Q) \parallel (y.P \; |!x : A| \; R)\} \vdash_\eta \Delta; \Gamma$ ((3) and (9))

Case: [D-C!Sh] right-to-left,

$$\mathsf{share} \; z \; \{(y.P \; |!x : A| \; Q) \parallel (y.P \; |!x : A| \; R)\} \equiv y.P \; |!x : A| \; \mathsf{share} \; z \; \{Q \parallel R\}$$

(1) $\Delta = \Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{Y}} \; B$ (2) $y.P \; |!x : A| \; Q \vdash_\eta \Delta_1, z : \mathbf{U}_{\mathcal{Y}_1} \; B; \Gamma$
(3) $y.P \; |!x : A| \; R \vdash_\eta \Delta_2, z : \mathbf{U}_{\mathcal{Y}_2} \; B; \Gamma$
(4) $\mathcal{Y}_1 \oplus \mathcal{Y}_2 = \mathcal{Y}$, for some $\Delta_1, \Delta_2, B, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}$
([Tsh^{-1}] and $\mathsf{share} \; z \; \{(y.P \; |!x : A| \; Q) \parallel (y.P \; |!x : A| \; R)\} \vdash_\eta \Delta; \Gamma$)
(5) $P \vdash_\eta y : A; \Gamma$ (6) $Q \vdash_\eta \Delta_1, z : \mathbf{U}_{\mathcal{Y}_1} \; B; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (2))
(7) $R \vdash_\eta \Delta_2, z : \mathbf{U}_{\mathcal{Y}_2} \; B; \Gamma, x : \overline{A}$ ([Tcut!$^{-1}$] and (3))

(8) $\mathsf{share}\ z\ \{Q \mathbin{||} R\} \vdash_\eta \Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{Y}}\ B; \Gamma, x : \overline{A}$ ([Tsh], (6), (7) and (4))
(9) $y.P\ |!x : A|\ (\mathsf{share}\ z\ \{Q \mathbin{||} R\}) \vdash_\eta \Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{Y}}\ B; \Gamma$ ([Tcut!], (5) and (8))
(10) $y.P\ |!x : A|\ (\mathsf{share}\ z\ \{Q \mathbin{||} R\}) \vdash_\eta \Delta; \Gamma$ ((1) and (9))

Case: [ShRel] $\mathsf{share}\ x\ \{\mathsf{release}\ x \mathbin{||} P\} \leq P$.
(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $\mathsf{release}\ x \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $P \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{\mathsf{release}\ x \mathbin{||} P\} \vdash_\eta \Delta; \Gamma$)
(5) $\Delta_1 = \emptyset$ (6) $\mathcal{X}_1 = f$ ([Tfree^{-1}] and (2))
(7) $P \vdash_\eta \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ ((3) and (5))
(8) $\mathcal{X} = \mathcal{X}_2$ ((4) and (6))
(9) $P \vdash_\eta \Delta; \Gamma$ ((1), (7) and (8))

Case: [ShTake],

$$\mathsf{share}\ x\ \{\mathsf{take}\ x(y_1); P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2)\} \\ \leq \mathsf{take}\ x(y_1); \mathsf{share}\ x\ \{P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2\}$$

(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $\mathsf{take}\ x(y_1); P_1 \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $\mathsf{take}\ x(y_2); P_2 \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $\Delta_1, \Delta_2, A, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{\mathsf{take}\ x(y_1); P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2\} \vdash_\eta \Delta; \Gamma$)
(5) $P_1 \vdash_\eta \Delta_1, x : \mathsf{U}_\circ A, y_1 : \vee A; \Gamma$ (6) $\mathcal{X}_1 = f$, $\mathcal{X}_2 = e$ ([Ttake^{-1}] and (2) and (3))
(6) $\mathsf{share}\ x\ \{P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2\} \vdash_\eta \Delta_1, \Delta_2, x : \mathsf{U}_\circ A, y_1 : \vee A; \Gamma$ ([Tsh],(5), (4) and (6))
(7) $\mathsf{take}\ x(y_1); \mathsf{share}\ x\ \{P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2\} \vdash_\eta \Delta_1, \Delta_2, x : \mathsf{U}_\bullet A; \Gamma$ ([Ttake] and (6))
(8) $\mathsf{take}\ x(y_1); \mathsf{share}\ x\ \{P_1 \mathbin{||} \mathsf{take}\ x(y_2); P_2\} \vdash_\eta \Delta; \Gamma$ ((7), (1) and (6))

Case: [ShPut], $\mathsf{share}\ x\ \{\mathsf{put}\ x(y.P); Q \mathbin{||} R\} \leq \mathsf{put}\ x(y.P); \mathsf{share}\ x\ \{Q \mathbin{||} R\}$.
(1) $\Delta = \Delta_1, \Delta_2, x : \mathbf{U}_{\mathcal{X}}\ A$ (2) $\mathsf{put}\ x(y.P); Q \vdash_\eta \Delta_1, x : \mathbf{U}_{\mathcal{X}_1}\ A; \Gamma$
(3) $R \vdash_\eta \Delta_2, x : \mathbf{U}_{\mathcal{X}_2}\ A; \Gamma$ (4) $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$, for some $A, \Delta_1, \Delta_2, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$
([Tsh^{-1}] and $\mathsf{share}\ x\ \{\mathsf{put}\ x(y.P); Q \mathbin{||} R\} \vdash_\eta \Delta; \Gamma$)
(5) $\mathcal{X}_1 = e$ (6) $\Delta_1 = \Delta_{11}, \Delta_{12}$ (7) $P \vdash_\eta \Delta_{11}, y : \wedge\overline{A}; \Gamma$ (8) $Q \vdash_\eta \Delta_{12}, x : \mathsf{U}_\bullet A; \Gamma$ ([Tput^{-1}] and (2))
(9) $\mathcal{X}_2 = f$ (10) $\mathcal{X} = e$ ((4) and (5))
(10) $\mathsf{share}\ x\ \{Q \mathbin{||} R\} \vdash_\eta \Delta_{12}, \Delta_2, x : \mathsf{U}_\bullet A; \Gamma$ ([Tsh], (8), (3), (9) and $f \oplus f = f$)
(11) $\mathsf{put}\ x(y.P); \mathsf{share}\ x\ \{Q \mathbin{||} R\} \vdash_\eta \Delta_{11}, \Delta_{12}, \Delta_2, x : \mathsf{U}_\circ A; \Gamma$ ([Tput], (7) and (10))
(12) $\Delta_{11}, \Delta_{12}, \Delta_2, x : \mathsf{U}_\circ A = \Delta$ ((1), (6) and (10))
(13) $\mathsf{put}\ x(y.P); \mathsf{share}\ x\ \{Q \mathbin{||} R\} \vdash_\eta \Delta; \Gamma$ ((11) and (12))

Case: [0M] left-to-right, $P \mathbin{||} \mathsf{0} \equiv P$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $P \vdash_\eta \Delta_1; \Gamma$ (3) $\mathsf{0} \vdash_\eta \Delta_3; \Gamma$, for some Δ_1, Δ_2
([Tmix^{-1}] and $P \mathbin{||} \mathsf{0} \vdash_\eta \Delta; \Gamma$)
(4)$\Delta_3 = \emptyset$ ([T0^{-1}] and (3))
(5)$\Delta = \Delta_1$ ((1) and (4))
(6)$P \vdash_\eta \Delta; \Gamma$ ((2) and (5))

Case: [0M] right-to-left, $P \equiv P \mathbin{||} \mathsf{0}$.

(1) $0 \vdash_\eta \emptyset; \Gamma$ ([T0])
(2) $P \parallel 0 \vdash_\eta \Delta; \Gamma$ ([Tmix], $P \vdash_\eta \Delta; \Gamma$ and (1))

Theorem B.2 (Type Preservation $\rightarrow$). *If $P \vdash_\eta \Delta; \Gamma$ and $P \rightarrow Q$, then $Q \vdash_\eta \Delta; \Gamma$.*

Proof. By induction on a derivation tree for $P \rightarrow Q$ and case analysis on the root rule.

Case: [fwd], $\mathsf{fwd}\ x\ y\ |y : A|\ P \rightarrow \{x/y\}P$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{fwd}\ x\ y \vdash_\eta \Delta_1, y : A; \Gamma$ (3) $P \vdash_\eta \Delta_2, y : \overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $\mathsf{fwd}\ x\ y\ |y : A|\ P \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1, y : A = x : \overline{B}, y : B$, for some B ([Tfwd^{-1}] and (2))
(5) $\Delta_1 = x : \overline{A}$ and $A = B$ ((4))
(6) $\{x/y\}P \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$ (Lemma B.4(1) and (3))
(7) $\{x/y\}P \vdash_\eta \Delta_2, \Delta_1; \Gamma$ ((5) and (6))
(8) $\{x/y\}P \vdash_\eta \Delta; \Gamma$ ((1) and (7))

Case: $[\mathbf{1}\bot]$, $\mathsf{close}\ x\ |x : \mathbf{1}|\ \mathsf{wait}\ x; P \rightarrow P$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{close}\ x \vdash_\eta \Delta_1, x : \mathbf{1}; \Gamma$ (3) $\mathsf{wait}\ x; P \vdash_\eta \Delta_2, x : \bot; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $\mathsf{close}\ x\ |x : \mathbf{1}|\ \mathsf{wait}\ x; P \vdash_\eta \Delta; \Gamma$)
(3) $\Delta_1 = \emptyset$ ([T$\mathbf{1}^{-1}$] and (2))
(4) $P \vdash_\eta \Delta_2; \Gamma$ ([T$\bot^{-1}$] and (3))
(5) $P \vdash_\eta \Delta; \Gamma$ ((1), (3) and (4))

Case: $[\otimes\invamp]$, $\mathsf{send}\ x(y.P); Q\ |x : A \otimes B|\ \mathsf{recv}\ x(z); R \rightarrow Q\ |x : B|\ (P\ |y : A|\ \{y/z\}R)$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{send}\ x(y.P); Q \vdash_\eta \Delta_1, x : A \otimes B; \Gamma$ (3) $\mathsf{recv}\ x(z); R \vdash_\eta \Delta_2, x : \overline{A} \invamp \overline{B}; \Gamma$
for some Δ_1, Δ_2 ([Tcut^{-1}] and $\mathsf{send}\ x(y.P); Q\ |x : A \otimes B|\ \mathsf{recv}\ x(z); R \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1 = \Delta_{11}, \Delta_{12}$ (5) $P \vdash_\eta \Delta_{11}, y : A; \Gamma$ (6) $Q \vdash_\eta \Delta_{12}, x : B; \Gamma$, for some Δ_{11}, Δ_{12} ([T$\otimes^{-1}$] and (2))
(7) $R \vdash_\eta \Delta_2, z : \overline{A}, x : \overline{B}; \Gamma$ ([T$\invamp^{-1}$] and (3))
(8) $\{y/z\}R \vdash_\eta \Delta_2, y : \overline{A}, x : \overline{B}; \Gamma$ (Lemma B.4(1) and (7))
(9) $P\ |y : A|\ \{y/z\}R \vdash_\eta \Delta_{11}, \Delta_2, x : \overline{B}; \Gamma$ ([Tcut], (5) and (8))
(10) $Q\ |x : B|\ (P\ |y : A|\ \{y/z\}R) \vdash_\eta \Delta_{12}, \Delta_{11}, \Delta_2; \Gamma$ ([Tcut], (6) and (9))
(11) $Q\ |x : B|\ (P\ |y : A|\ \{y/z\}R) \vdash_\eta \Delta; \Gamma$ ((1), (4) and (10))

Case: $[\&\oplus_l]$, $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x : A \& B|\ x.\mathsf{inl}; R \rightarrow P\ |x : A|\ R$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\} \vdash_\eta \Delta_1, x : A \& B; \Gamma$
(3) $x.\mathsf{inl}; R \vdash_\eta \Delta_2, x : \overline{A} \oplus \overline{B}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x : A \& B|\ x.\mathsf{inl}; R \vdash_\eta \Delta; \Gamma$
(4) $P \vdash_\eta \Delta_1, x : A$ (5) $Q \vdash_\eta \Delta_1, x : B; \Gamma$ ([T$\&^{-1}$] and (2))
(6) $R \vdash_\eta \Delta_2, x : \overline{A}; \Gamma$ (T$\oplus_l^{-1}$] and (3))
(7) $P\ |x : A|\ R \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (4) and (6))
(8) $P\ |x : A|\ R \vdash_\eta \Delta; \Gamma$ ((1) and (7))

Case: $[\&\oplus_r]$, $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x : A \mathbin{\&} B|\ x.\mathsf{inr}; R \to Q\ |x : B|\ R$.

(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\} \vdash_\eta \Delta_1, x : A \mathbin{\&} B; \Gamma$

(3) $x.\mathsf{inr}; R \vdash_\eta \Delta_2, x : \overline{A} \oplus \overline{B}; \Gamma$, for some Δ_1, Δ_2

(Tcut^{-1}] and $\mathsf{case}\ x\ \{|\mathsf{inl} : P,\ |\mathsf{inr} : Q\}\ |x : A \mathbin{\&} B|\ x.\mathsf{inl}; R \vdash_\eta \Delta; \Gamma$

(4) $P \vdash_\eta \Delta_1, x : A$ (5) $Q \vdash_\eta \Delta_1, x : B; \Gamma$ ([T$\&^{-1}$] and (2))

(6) $R \vdash_\eta \Delta_2, x : \overline{B}; \Gamma$ ([T$\oplus_r^{-1}$] and (3))

(7) $Q\ |x : B|\ R \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (5) and (6))

(8) $P\ |x : A|\ R\ \vdash_\eta \Delta; \Gamma$ ((1) and (7))

Case: $[!?]$, $!x(y); P\ |x :!A|\ ?x; Q \to y.P\ |!x : A|\ Q$.

(1) $\Delta = \Delta_1, \Delta_2$ (2) $!x(y); P \vdash_\eta \Delta_1, x :!A; \Gamma$

(3) $?x; Q \vdash_\eta \Delta_2, x :?\overline{A}; \Gamma$, for some Δ_1, Δ_2

([Tcut^{-1}] and $!x(y); P\ |x :!A|\ ?x; Q \vdash_\eta \Delta; \Gamma$

(4) $\Delta_1 = \emptyset$ (5) $P \vdash_\eta y : A; \Gamma$ ([T!$^{-1}$] and (2))

(6) $Q \vdash_\eta \Delta_2; \Gamma, x : \overline{A}$ ([T?$^{-1}$] and (3))

(7) $y.P\ |!x : A|\ Q\ \vdash_\eta \Delta_2; \Gamma$ ([Tcut!], (5) and (6))

(8) $y.P\ |!x : A|\ Q\ \vdash_\eta \Delta; \Gamma$ ((1), (4) and (7))

Case: [call], $y.P\ |!x : A|\ \mathsf{call}\ x(z); Q \to \{z/y\}P\ |z : A|\ (y.P\ |!x : A|\ Q)$.

(1) $P \vdash_\eta y : A; \Gamma$ (2) $\mathsf{call}\ x(z); Q \vdash_\eta \Delta; \Gamma, x : \overline{A}$

([Tcut!$^{-1}$] and $y.P\ |!x : A|\ \mathsf{call}\ x(z); Q \vdash_\eta \Delta; \Gamma$

(3) $Q \vdash_\eta \Delta, z : \overline{A}; \Gamma, x : \overline{A}$ ([Tcall^{-1}] and (2))

(4) $y.P\ |!x : A|\ Q \vdash_\eta \Delta, z : \overline{A}; \Gamma$ ([Tcut!], (1) and (3))

(5) $\{z/y\}P\ \vdash_\eta z : A; \Gamma$ (Lemma B.4(1) and (1))

(6) $\{z/y\}P\ |z : A|\ (y.P\ |!x : A|\ Q) \vdash_\eta \Delta; \Gamma$ ([Tcut], (5) and (4))

Case: $[\exists\forall]$, $\mathsf{sendty}\ x(A); P\ |x : \exists X.B|\ \mathsf{recvty}\ x(X); Q \to P\ |x : \{A/X\}B|\ \{A/X\}Q$.

(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{sendty}\ x(A); P \vdash_\eta \Delta_1, x : \exists X.B; \Gamma$

(3) $\mathsf{recvty}\ x(X); Q \vdash_\eta \Delta_2, x : \forall X.\overline{B}; \Gamma$, for some Δ_1, Δ_2

([Tcut^{-1}] and $\mathsf{sendty}\ x(A); P\ |x : \exists X.B|\ \mathsf{recvty}\ x(X); Q \vdash_\eta \Delta; \Gamma$

(4) $P \vdash_\eta \Delta_1, x : \{A/X\}B; \Gamma$ ([T$\exists^{-1}$] and (2))

(5) $Q \vdash_\eta \Delta_2, x : \overline{B}; \Gamma$ ([T$\forall^{-1}$] and (3))

(6) $\{A/X\}Q \vdash_\eta \Delta_2, x : \{A/X\}\overline{B}; \Gamma$ (Lemma B.4(2) and (5))

(7) $\{A/X\}Q \vdash_\eta \Delta_2, x : \overline{\{A/X\}B}; \Gamma$ ($\overline{\{A/X\}B} = \{A/X\}\overline{B}$ and (6))

(8) $P\ |x : \{A/X\}B|\ \{A/X\}Q \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (4) and (7))

(9) $P\ |x : \{A/X\}B|\ \{A/X\}Q \vdash_\eta \Delta; \Gamma$ ((1) and (8))

Case: $[\mu\nu]$, $\mathsf{unfold}_\mu\ x; P\ |x : \mu X.\ A|\ \mathsf{unfold}_\nu\ x; Q \to P\ |x : \{\mu X.\ A/X\}A|\ Q$.

(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{unfold}_\mu\ x; P \vdash_\eta \Delta_1, x : \mu X.\ A; \Gamma$

(3)$\mathsf{unfold}_\nu\ x; Q \vdash_\eta \Delta_2, x : \nu X.\ \{\overline{X}/X\}\overline{A}; \Gamma$, for some Δ_1, Δ_2

([Tcut^{-1}] and $\mathsf{unfold}_\mu\ x; P\ |x : \mu X.\ A|\ \mathsf{unfold}_\nu\ x; Q \vdash \Delta; \Gamma$)

(4) $P \vdash_\eta \Delta_1, x : \{\mu X.\ A/X\}A; \Gamma$ ([Tμ^{-1}] and (2))

(5) $Q \vdash_\eta \Delta_2, x : \{\nu X.\ \{\overline{X}/X\}\overline{A}/X\}(\{\overline{X}/X\}\overline{A}); \Gamma$ ([Tν^{-1}] and (3))

(6) $Q \vdash_\eta \Delta_2, x : \{\mu X.\ A/X\}\overline{A}; \Gamma$ ((5) and (*)

(7) $P\ |x : \{\mu X.\ A/X\}A|\ Q \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (5) and (6))

(8) $P \mathbin{|x : \{\mu X.\ A/X\}A|} Q \vdash_\eta \Delta; \Gamma$ ((7) and (1))

To obtain (*):

$$\begin{aligned}\{\nu X.\ \{\overline{X}/X\}\overline{A}/X\}(\{\overline{X}/X\}\overline{A}) &= \{\overline{\nu X.\ \{\overline{X}/X\}\overline{A}}/X\}\overline{A}) \\ &= \{(\mu X.\ \{\overline{X}/X\}\overline{\{\overline{X}/X\}\overline{A}})/X\}\overline{A} \\ &= \{(\mu X.\ \{\overline{X}/X\}\{\overline{X}/X\}\overline{\overline{A}})/X\}\overline{A}) \\ &= \{(\mu X.\ \{\overline{X}/X\}\{\overline{X}/X\}A)/X\}\overline{A}) \\ &= \{\mu X.\ A/X\}\overline{A}\end{aligned}$$

Case: [corec],

$$\begin{aligned}&\mathsf{unfold}_\mu\ x; P \mathbin{|x : \mu X.\ A|} \mathsf{corec}\ Y(z, \vec{w}); Q\ [x, \vec{y}] \\ &\quad \to P \mathbin{|x : \{\mu X.\ A/X\}A|} \sigma(\{\mathsf{corec}\ Y(z, \vec{w}); Q/Y\}Q)\end{aligned}$$

where σ is the substitution map given by $\sigma = \{x/z\}\{\vec{y}/\vec{w}\}$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{unfold}_\mu\ x; P \vdash_\eta \Delta_1, x : \mu X.\ A; \Gamma$
(3) $\mathsf{corec}\ Y(z, \vec{w}); Q\ [x, \vec{y}] \vdash_\eta \Delta_2, x : \nu X.\ \{\overline{X}/X\}\overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $\mathsf{unfold}_\mu\ x; P \mathbin{|x : \mu X.\ A|} \mathsf{corec}\ Y(x, \vec{y}); Q \vdash \Delta; \Gamma$)
(4) $P \vdash_\eta \Delta_1, x : \{\mu X.\ A/X\}A; \Gamma$ ([Tμ^{-1}] and (2))
(5) $\eta' = \eta, Y(z, \vec{w}) \mapsto \sigma^{-1}(\Delta_2, z : X; \Gamma)$ (6) $Q \vdash_{\eta'} \sigma^{-1}(\Delta_2, z : \{\overline{X}/X\}\overline{A}; \Gamma)$
([Tloop^{-1}] and (3))
(7) $\{\mathsf{corec}\ Y(z, \vec{w}); Q/Y\}Q \vdash_\eta \sigma^{-1}(\Delta_2, x : \{\nu X.\ \{\overline{X}/X\}\overline{A}/X\}(\{\overline{X}/X\}\overline{A}); \Gamma)$
(Lemma B.4(3), (3), (5) and (6))
(8) $\sigma(\{\mathsf{corec}\ Y(z, \vec{w}); Q/Y\}Q) \vdash_\eta \Delta_2, x : \{\nu X.\ \{\overline{X}/X\}\overline{A}/X\}(\{\overline{X}/X\}\overline{A}); \Gamma$
((7) and since σ^{-1} is the inverse of σ
(9) $\sigma(\{\mathsf{corec}\ Y(x, \vec{y}); Q/Y\}Q) \vdash_\eta \Delta_2, x : \{\mu X.\ A/X\}A; \Gamma$ ((8) and (*) from case [$\mu\nu$] above)
(10) $P \mathbin{|x : \{\mu X.\ A/X\}A|} \{\mathsf{corec}\ Y(x, \vec{y}); Q/Y\}Q \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (4) and (9))
(11) $P \mathbin{|x : \{\mu X.\ A/X\}A|} \{\mathsf{corec}\ Y(x, \vec{y}); Q/Y\}Q \vdash_\eta \Delta; \Gamma$ ((1) and (10))

Case: [$\wedge\vee$d], $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \mathbin{|a : \wedge A|} \mathsf{discard}\ a \to \mathsf{discard}\ \vec{b} \mathbin{||} \mathsf{release}\ \vec{c}$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \vdash_\eta \Delta_1, v : \wedge A; \Gamma$
(3) $\mathsf{discard}\ a \vdash_\eta \Delta_2, v : \vee\overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \mathbin{|a : \wedge A|} \mathsf{discard}\ a \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1 = \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}$ (5) $P \vdash_\eta \Delta_1, a : A; \Gamma$, for some $\vec{b}, \vec{B}, \vec{c}, \vec{C}$
([Taffine^{-1}] and (2))
(6) $\Delta_2 = \emptyset$ ([Tdiscard^{-1}] and (3))
(7) $\mathsf{discard}\ \vec{b} \mathbin{||} \mathsf{release}\ \vec{c} \vdash_\eta \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}; \Gamma$ ([Tmix], [Tdiscard] and [Trelease])
(8) $\vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C} = \Delta$ ((1), (4) and (6))
(9) $\mathsf{discard}\ \vec{b} \mathbin{||} \mathsf{release}\ \vec{c} \vdash_\eta \Delta; \Gamma$ ((7) and (8))

Case: [$\wedge\vee$u], $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \mathbin{|a : \wedge A|} \mathsf{use}\ a; Q \to P \mathbin{|a : A|} Q$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \vdash_\eta \Delta_1, v : \wedge A; \Gamma$
(3) $\mathsf{use}\ a; Q \vdash_\eta \Delta_2, v : \vee\overline{A}; \Gamma$, for some Δ_1, Δ_2
([Tcut^{-1}] and $\mathsf{affine}_{\vec{b},\vec{c}}\ a; P \mathbin{|a : \wedge A|} \mathsf{use}\ a; Q \vdash_\eta \Delta; \Gamma$

(4) $\Delta_1 = \vec{b} : \vee\vec{B}, \vec{c} : \mathsf{U}_\bullet\vec{C}$ (5) $P \vdash_\eta \Delta_1, a : A; \Gamma$, for some $\vec{b}, \vec{B}, \vec{c}, \vec{C}$ ([Taffine^{-1}] and (2))
(6) $Q \vdash_\eta \Delta_2, a : \overline{A}; \Gamma$ ([Tuse^{-1}] and (3))
(7) $P \;|a : A|\; Q \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (5) and (6))
(8) $P \;|a : A|\; Q \vdash_\eta \Delta; \Gamma$ ((1) and (7))

Case: $[\mathsf{S}_\bullet\mathsf{U}_\bullet\mathrm{f}]$, $\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; \mathsf{release}\ c \to P \;|a : \wedge A|\; \mathsf{discard}\ a$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{cell}\ c(a.P) \vdash_\eta \Delta_1, c : \mathsf{S}_\bullet A; \Gamma$ (3) $\mathsf{release}\ c \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet\overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; \mathsf{release}\ c \vdash_\eta \Delta; \Gamma$)
(4) $P \vdash_\eta \Delta_1, a : \wedge A; \Gamma$ ([Tcell^{-1}] and (2))
(5) $\Delta_2 = \emptyset$ ([Tfree^{-1}] and (3))
(6) $\mathsf{discard}\ a \vdash_\eta a : \vee\overline{A}; \Gamma$ ([Tdiscard])
(7) $P \;|a : \wedge A|\; \mathsf{discard}\ a \vdash_\eta \Delta_1; \Gamma$ ([Tcut], (4) and (6))
(8) $\Delta = \Delta_1$ ((1) and (5))
(9) $P \;|a : \wedge A|\; \mathsf{discard}\ a \vdash_\eta \Delta; \Gamma$ ((7) and (8))

Case: $[\mathsf{S}_\bullet\mathsf{U}_\bullet\mathrm{t}]$,

$$\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; \mathsf{take}\ c(a'); Q \to \{a'/a\}P \;|a' : \wedge A|\; (\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; Q)$$

(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{cell}\ c(a.P) \vdash_\eta \Delta_1, c : \mathsf{S}_\bullet A; \Gamma$
(3) $\mathsf{take}\ c(a'); Q \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet\overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; \mathsf{take}\ c(a'); Q \vdash_\eta \Delta; \Gamma$)
(4) $P \vdash_\eta \Delta_1, a : \wedge A; \Gamma$ ([Tcell^{-1}] and (2))
(5) $Q \vdash_\eta \Delta_2, a : \vee\overline{A}, c : \mathsf{U}_\circ\overline{A}; \Gamma$ ([Ttake^{-1}] and (3))
(6) $\mathsf{empty}\ c \vdash_\eta c : \mathsf{S}_\circ A; \Gamma$ ([Tempty])
(7) $\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; Q \vdash_\eta \Delta_2, a : \vee\overline{A}; \Gamma$ ([Tcut], (6) and (5))
(8) $\{a'/a\}P \vdash_\eta \Delta_1, a :' \wedge A; \Gamma$ (Lemma B.2(Tsubs) and (4))
(9) $\{a'/a\}P \;|a' : \wedge A|\; (\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; Q) \vdash_\eta \Delta_1, \Delta_2; \Gamma$ ([Tcut], (8) and (7))
(10) $\{a'/a\}P \;|a' : \wedge A|\; (\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; Q) \vdash_\eta \Delta; \Gamma$ ((1) and (9))

Case: $[\mathsf{S}_\circ\mathsf{U}_\circ]$, $\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; \mathsf{put}\ c(a.P); Q \to \mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; Q$.
(1) $\Delta = \Delta_1, \Delta_2$ (2) $\mathsf{empty}\ c \vdash_\eta \Delta_1, c : \mathsf{S}_\circ A; \Gamma$ (3) $\mathsf{put}\ c(a.P); Q \vdash_\eta \Delta_2, c : \mathsf{U}_\circ\overline{A}; \Gamma$, for some Δ_1, Δ_2 ([Tcut^{-1}] and $\mathsf{empty}\ c \;|c : \mathsf{S}_\circ A|\; \mathsf{put}\ c(a.P); Q \vdash_\eta \Delta; \Gamma$)
(4) $\Delta_1 = \emptyset$ ([Tempty^{-1}] and (2))
(5) $\Delta_2 = \Delta_{21}, \Delta_{22}$ (6) $P \vdash_\eta \Delta_{21}, a : \wedge A; \Gamma$ (7) $Q \vdash_\eta \Delta_{22}, c : \mathsf{U}_\bullet\overline{A}; \Gamma$ ([Tput^{-1}] and (3))
(8) $\mathsf{cell}\ c(a.P) \vdash_\eta \Delta_{21}, c : \mathsf{S}_\bullet A; \Gamma$ ([Tcell] and (6))
(9) $\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; Q \vdash_\eta \Delta_{21}, \Delta_{22}; \Gamma$ ([Tcut], (8) and (7))
(10) $\Delta = \Delta_{21}, \Delta_{22}$ ((1), (4) and (5))
(11) $\mathsf{cell}\ c(a.P) \;|c : \mathsf{S}_\bullet A|\; Q \vdash_\eta \Delta; \Gamma$ ((9) and (10))

Case: $[\leq]$, $P \leq P'$ and $P' \to Q'$ and $Q' \leq Q \;\supset\; P \to Q$.
(1) $P' \vdash_\eta \Delta; \Gamma$ (Theorem B.1, $P \vdash_\eta \Delta; \Gamma$ and $P \leq P'$)
(2) $Q' \vdash_\eta \Delta; \Gamma$ (i.h., (1) and $P' \to Q'$)

(3) $Q \vdash_\eta \Delta; \Gamma$ (Theorem B.1, (2) and $Q' \leq P$)

Case: [cong], $P \to Q \supset \mathcal{C}[P] \to \mathcal{C}[Q]$.
(1) $P \vdash_\eta \Delta'; \Gamma'$, for some Δ', Γ' (Lemma B.1 and $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$)
(2) $Q \vdash_\eta \Delta'; \Gamma'$ (i.h., (1) and $P \to Q$)
(3) $\mathcal{C}[Q] \vdash_\eta \Delta; \Gamma$ (Lemma B.1 , (1), (2) and $\mathcal{C}[P] \vdash_\eta \Delta; \Gamma$)

C Progress

We prove that CLASS enjoys the progress property (Theorem C.1), namely that all closed live processes reduce. Progress is a liveness property: it guarantees that closed live processes will never get stuck.

C.1 Live Processes.

We start by defining what means for a process to be live (Definition C.1).

Definition C.1 (Live Process). *A process P is live if $P = \mathcal{C}[\mathcal{A}]$ or $P = \mathcal{C}[\mathsf{fwd}\ x\ y]$ for some static context $\mathcal{C}$ and action $\mathcal{A}$.*

Intuitively, a process is live if it presents an unguarded action or forwarder waiting to interact, that action lies only under the scope of a static construct (mix, linear or unrestricted cut or share). As a consequence of our linear typing discipline, all the typed processes $P \vdash_\eta \Delta; \Gamma$ that (i) type with a nonempty linear context Δ and (ii) with an empty map η are necessarily live, as established by the following lemma. The latter condition (ii) is necessary so as to exclude processes variables $X(\vec{y})$ since they offer no structure for interaction, they are not live.

Lemma C.1. *If $P \vdash_\emptyset \Delta; \Gamma$ and $\Delta \neq \emptyset$, then P is live.*

Proof. By induction on a derivation of $P \vdash_\emptyset \Delta; \Gamma$. Case [T0] holds vacuously because it types inaction $\mathbf{0}$ with an empty linear context. Case [Tvar] holds vacuously because it types a variable with a nonempty recursion map η.

Cases which introduce the forwarder construct or an action hold trivially since P can be written as $-[\mathsf{fwd}\ x\ y]$ or $-[\mathcal{A}]$, where $-$ is the empty static process context and $\mathcal{A}$ is an action.

The remaining cases are [Tmix], [Tcut], [Tcut!], [Tsh], [TshL] and [TshR]. In these cases, from the fact that the conclusion types with a nonempty linear context we can infer that at least one of the premisses types with a nonempty linear context as well, so that we can apply the inductive hypotheses to infer liveness of one of the arguments of P, which then implies liveness of P. We illustrate with cases [Tmix] and [Tsh].

Case [Tmix]
We have

$$\dfrac{\dfrac{\vdots}{P_1 \vdash \Delta_1; \Gamma} \quad \dfrac{\vdots}{P_2 \vdash \Delta_2; \Gamma}}{P_1 \mid\mid P_2 \ \vdash \Delta_1, \Delta_2; \Gamma} \ [\text{Tmix}]$$

where $P = P_1 \mid\mid P_2$ and $\Delta = \Delta_1, \Delta_2$.
Since $\Delta \neq \emptyset$, then either $\Delta_1 \neq \emptyset$ or $\Delta_2 \neq \emptyset$.
Assume w.l.o.g. that $\Delta_1 \neq \emptyset$.
By applying the i.h. to $P_1 \vdash \Delta_1; \Gamma$ we conclude that $P_1 = \mathcal{C}_1[\mathcal{X}]$, where $\mathcal{C}$ is a static context and $\mathcal{X}$ is either an action or a forwarder.
Let $\mathcal{C} = \mathcal{C}_1 \mid\mid P_2$. Then, $\mathcal{C}$ is static and $P = \mathcal{C}[\mathcal{X}]$.

Case [Tsh].
We have

$$\dfrac{\dfrac{\vdots}{P_1 \vdash \Delta_1, x : \mathsf{U}_\bullet A; \Gamma} \quad \dfrac{\vdots}{P_2 \vdash \Delta_2, x : \mathsf{U}_\bullet A; \Gamma}}{\mathsf{share}\ x\ \{P_1 \mid\mid P_2\} \ \vdash \Delta_1, \Delta_2, x : \mathsf{U}_\bullet A; \Gamma} \ [\text{Tsh}]$$

where $P = \mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ and $\Delta = \Delta_1, \Delta_2, x : \mathsf{U}_\bullet A$.
By applying the i.h. to $P_1 \vdash \Delta_1, x : \mathsf{U}_\bullet A; \Gamma$ we conclude that $P_1 = \mathcal{C}_1[\mathcal{Y}]$, where $\mathcal{C}$ is a static context and $\mathcal{Y}$ is either an action or a forwarder.
Let $\mathcal{C} = \mathsf{share}\ x\ \{\mathcal{C}_1 \mid\mid P_2\}$. Then, $\mathcal{C}$ is static and $P = \mathcal{C}[\mathcal{Y}]$.
Notice that in this case both premisses type with a nonempty linear context, independently of the conclusion, and so the hypothesis that Δ is nonempty is superfluous. We could have opted to establish liveness of $\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ by applying the i.h. to $P_2 \vdash \Delta_2, x : \mathsf{U}_\bullet A; \Gamma$ instead. A similar situation happens for [Tcut].

C.2 Observability Predicate and Properties

The progress Theorem C.1 states that a closed, i.e. typed with an empty typing context $P \vdash_\emptyset \emptyset; \emptyset$ and empty map η, and live process P reduces. If one tries to prove this statement by induction on a typing derivation for $P \vdash_\emptyset \emptyset; \emptyset$ one soon realises, when analysing the case [Tcut], that we need to say something about open processes. That is, to compositionally prove progress we need to characterise the potential interactions of (possibly open) typed processes, for which we define the following observability predicate, which is akin to π-calculus observability (cf. [64]). Our proof is along the lines of [21], but here we rely in an observability predicated, whereas in [21] progress is established by relying on a labelled transition system instead.

Definition C.2 (Observability Predicate). *The relation $P \downarrow_{x:\sigma}$, where $\sigma = fwd$ or $\sigma = act$, is defined by the rules of Figure 24. We say that x is an observable*

$$\frac{}{\mathsf{fwd}\ x\ y\downarrow_{x:\mathrm{fwd}}}\ [\mathrm{fwd}] \quad \frac{s(\mathcal{A})=x}{\mathcal{A}\downarrow_{x:\mathrm{act}}}\ [\mathrm{act}]$$

$$\frac{P\downarrow_{x:\sigma}}{(P \mid\mid Q)\downarrow_{x:\sigma}}\ [\mathrm{mix}] \quad \frac{P\downarrow_{y:\sigma} \quad y\neq x}{(P \mid x\mid Q)\downarrow_{y:\sigma}}\ [\mathrm{cut}] \quad \frac{Q\downarrow_{z:\sigma} \quad z\neq x}{(y.P \mid !x\mid Q)\downarrow_{z:\sigma}}\ [\mathrm{cut!}]$$

$$\frac{P\downarrow_{y:\sigma} \quad y\neq x}{(\mathsf{share}\ x\ \{P \mid\mid Q\})\downarrow_{y:\sigma}}\ [\mathrm{share}] \quad \frac{P\leq Q \quad Q\downarrow_{x:\sigma}}{P\downarrow_{x:\sigma}}\ [\leq]$$

Fig. 24: Observability Predicate $P\downarrow_{x:\sigma},\ \sigma\in\{\mathrm{fwd},\mathrm{act}\}$

of P or that we can observe x in P, written $P\downarrow_x$, if either $P\downarrow_{x:fwd}$ or $P\downarrow_{x:act}$. If $P\downarrow_{x:act}$, we say that x is an observable action of P. If $P\downarrow_{x:fwd}$, we say that x is an observable forwarder of P.

The definition of $P\downarrow_x$ is explicitly closed under $\leq$ (rule $[\leq]$) and propagates observations on the various static operators. For example, x is an observable of a mix $P \mid\mid Q$, provided x is an observable of one of its arguments P or Q. The same principle applies to the cut construct with the proviso that we can never observe the name x in a cut $P \mid x\mid Q$ since it is kept private to the interacting processes P and Q.

We can always observe the subject of an action (rule [act]) and we can observe the constituent names x, y of a forwarder $\mathsf{fwd}\ x\ y$: observation of x is direct from rule [fwd], whereas observation of y follows because of the $\equiv$ commuting rule [fwd] $\mathsf{fwd}\ x\ y \equiv \mathsf{fwd}\ y\ x$

$$\frac{\mathsf{fwd}\ x\ y\ \equiv \mathsf{fwd}\ y\ x \qquad \dfrac{}{\mathsf{fwd}\ y\ x\downarrow_y}\ [\mathrm{fwd}]}{\mathsf{fwd}\ x\ y\downarrow_x}\ [\equiv]$$

In a share $\mathsf{share}\ x\ \{P \mid\mid Q\}$, processes P and Q run concurrently freely communicating with the external context and sharing memory cell x. As a consequence, and similar to the cut construct, the share construct $\mathsf{share}\ x\ \{P \mid\mid Q\}$ propagates all the observations y for which $y\neq x$ (rule [share]).

Intuitively, x is an observable of a process P iff we can rewrite P in an $\leq$-equivalent form Q so as to expose an action with subject x or forwarder $\mathsf{fwd}\ x\ y$ and, furthermore, that action or forwarder in Q is not under the scope of a sharing construct on x.

We will now present some properties (Lemma C.2) concerning the observability predicate, which will play a key role to derive progress.

Lemma C.2 (Properties of $P\downarrow_x$). *The following properties hold*

(1) *Let $P\vdash_\eta \Delta, x:\mathsf{U}_\bullet A;\Gamma$ and $Q\vdash_\eta \Delta', x:\mathsf{U}_\bullet A;\Gamma$ be processes for which $P\downarrow_{x:act}$ and $Q\downarrow_{x:act}$. Then, $\mathsf{share}\ x\ \{P \mid\mid Q\}\downarrow_{x:act}$.*

(2) *Let $P\vdash_\eta \Delta, x:\mathsf{U}_\circ A;\Gamma$, $Q\vdash_\eta \Delta, x:\mathsf{U}_\bullet A;\Gamma$. If $P\downarrow_{x:act}$, then $\mathsf{share}\ x\ \{P \mid\mid Q\}\downarrow_{x:act}$.*

(3) *Let* $P \vdash_\eta \Delta, x : \mathsf{U}_\bullet A; \Gamma$, $Q \vdash_\eta \Delta, x : \mathsf{U}_\circ A; \Gamma$. *If* $Q \downarrow_{x:act}$, *then* $\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:act}$.
(4) *Let* $P \vdash_\eta \Delta, x : \overline{A}; \Gamma$ *and* $Q \vdash_\eta \Delta', x : A; \Gamma$ *be processes for which* $P \downarrow_{x:act}$ *and* $Q \downarrow_{x:act}$. *Then,* $P\ |x|\ Q$ *reduces.*
(5) *Let* $P \vdash_\eta \Delta, x : A; \Gamma$, $Q \vdash_\eta \Delta', x : A; \Gamma$ *be processes for which* $P \downarrow_{x:fwd}$. *Then,* $P\ |x|\ Q$ *reduces.*
(6) *Let* $P \vdash_\eta y : \overline{A}; \Gamma$ *and* $Q \vdash_\eta \Delta; \Gamma, x : A$ *be processes for which* $Q \downarrow_x$. *Then,* $y.P\ |!x|\ Q$ *reduces.*
(7) *Let* $P \vdash_\eta \Delta, x : A; \Gamma$ *and suppose that* $A \neq \mathsf{S}_\bullet B$ *and* $A \neq \mathsf{S}_\circ B$. *If* $P \downarrow_{x:fwd}$, *then either (i)* $P \downarrow_{y:fwd}$ *for some* $y : \overline{A} \in \Delta$ *or (ii)* P *reduces.*

Properties Lemma C.2(1)-(3) describe sufficient conditions to propagate observations x on a share $\mathsf{share}\ x\ \{P \mid\mid Q\}$.

Lemma C.2(1) states that we can observe a full usage on x in a $\mathsf{share}\ x\ \{P \mid\mid Q\}$ provided we can observe a full usage x on both P and Q. This full usage on x is propagated by applying either $\leq$ rule [RSh] or $\leq$ rule [TSh]. For example, by rule $\leq$ [RSh] we have $\mathsf{share}\ x\ \{\mathsf{release}\ x \mid\mid \mathsf{take}\ x(y); P\} \leq \mathsf{take}\ x(y); P$. Then

$$\frac{\mathsf{share}\ x\ \{\mathsf{release}\ x \mid\mid \mathsf{take}\ x(y); P\} \leq \mathsf{take}\ x(y); P \qquad \dfrac{s(\mathsf{take}\ x(y); P) = x}{\mathsf{take}\ x(y); P \downarrow_x}\,[\mathrm{act}]}{\mathsf{share}\ x\ \{\mathsf{release}\ x \mid\mid \mathsf{take}\ x(y); P\} \downarrow_x}\,[\leq]$$

Additionally, we can observe an empty usage x on $\mathsf{share}\ x\ \{P \mid\mid Q\}$ provided we can observe an empty usage x on either P or Q, as stated by Lemma C.2(2)-(3). The empty usage corresponds to a put action which can always be propagated to the top by applying $\leq$ rule [PSh].

Properties Lemma C.2(4)-(6) describe sufficient conditions for obtaining a reduction: either by observing two dual actions with subject x in a linear cut $P\ |x|\ Q$ (Lemma C.2(4)), by observing a forwarder x on a linear cut $P\ |x|\ Q$ (Lemma C.2(5)) or by observing a single action x in the right argument Q of an unrestricted cut $y.P\ |!x|\ Q$ (Lemma C.2(6)).

Lemma C.2(7) characterises the potential observation or reduction of a process that P for which $P \downarrow_{x:\mathrm{fwd}}$. Either name y occurs free, and P also offers a forwarder interaction at y, or lies in the scope of a cut $-\ |y|\ -$, in which case a reduction can be triggered (Lemma C.2(5)). The typing constraints $A \neq \mathsf{S}_\bullet B$ and $A \neq \mathsf{S}_\circ B$ exclude processes like $\mathsf{share}\ y\ \{\mathsf{fwd}\ x\ y \mid\mid Q\}$, that neither reduce nor offer an interaction at y. Intuitively, in this case, the share is suspended on the availability of cell usages at name y.

We prove properties Lemma C.2(1)-(7) of the observability predicate.

Lemma C.2(1) *Let* $P \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$ *and* $Q \vdash \Delta', x : \mathsf{U}_\bullet A; \Gamma$ *be processes for which* $P \downarrow_{x:act}$ *and* $Q \downarrow_{x:act}$. *Then,* $\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:act}$.

Proof. By double induction on derivation trees for $P \downarrow_{x:\mathrm{act}}$ and $Q \downarrow_{x:\mathrm{act}}$. For the base cases we apply either one of $\leq$ rules [RSh] or [TSh] in order to expose an observable action. For the inductive cases we consider that we are given a derivation tree for $P \downarrow_x$. This is w.l.o.g. since $\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv \mathsf{share}\ x\ \{Q \mid\mid P\}$.

For cases [mix], [cut], [cut!], [share] we commute the share on x with the principal form of P by applying either $\equiv$ rule [ShM], [CSh], [ShC!] or [ShSh].The inductive case [$\leq$] follows immediately because the relation $\leq$ is a congruence.

Case: The root rule of both $P \downarrow_{x:\text{act}}$ and $Q \downarrow_{x:\text{act}}$ is [act]. We have

$$\frac{s(\mathcal{A}) = x}{\mathcal{A} \downarrow_{x:\text{act}}}\ [\text{act}] \qquad \frac{s(\mathcal{B}) = x}{\mathcal{B} \downarrow_{x:\text{act}}}\ [\text{act}]$$

where $P = \mathcal{A}$ and $Q = \mathcal{B}$.
Since the subject of both actions $\mathcal{A}, \mathcal{B}$ - x - has the type $\mathsf{U}_\bullet A$ (in the linear typing context), we conclude that $\mathcal{A}$, $\mathcal{B}$ are either release or take actions.
Case: $\mathcal{A} = \mathsf{release}\ x$.
By applying $\leq$ rule [RSh] we obtain

$$\mathsf{share}\ x\ \{P \mid\mid Q\} = \mathsf{share}\ x\ \{\mathsf{release}\ x \mid\mid Q\} \leq Q$$

Hence

$$\frac{\mathsf{share}\ x\ \{P \mid\mid Q\} \leq Q \qquad \dfrac{\vdots}{Q \downarrow_{x:\text{act}}}}{\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:\text{act}}}\ [\leq]$$

Case: $\mathcal{B} = \mathsf{release}\ x$. Similar to case $\mathcal{A} = \mathsf{release}\ x$.
Case: $\mathcal{A} = \mathsf{take}\ x(y); P'$ and $\mathcal{B} = \mathsf{take}\ x(z); Q'$.
By applying $\leq$ rule [TSh] we obtain

$$\begin{aligned}&\mathsf{share}\ x\ \{\mathsf{take}\ x(y); P' \mid\mid \mathsf{take}\ x(z); Q'\} \ \leq \mathsf{take}\ x(y); R_1, \text{ where}\\ &\quad R_1 = \mathsf{share}\ x\ \{P' \mid\mid \mathsf{take}\ x(z); Q'\}\end{aligned}$$

Hence

$$\frac{\mathsf{share}\ x\ \{P \mid\mid Q\} \leq \mathsf{take}\ x(y); R_1 \qquad \dfrac{s(\mathsf{take}\ x(y); R_1) = x}{\mathsf{take}\ x(y); R_1 \downarrow_{x:\text{act}}}\ [\text{act}]}{\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:\text{act}}}\ [\leq]$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is [mix].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is [mix]. We have

$$\frac{P_1 \downarrow_{x:\text{act}}}{(P_1 \mid\mid P_2) \downarrow_{x:\text{act}}}\ [\text{mix}]$$

where $P = P_1 \mid\mid P_2$.
Since $P_1 \mid\mid P_2 \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ for which $P_1 \vdash \Delta_1, x : \mathsf{U}_\bullet A; \Gamma$ and $P_2 \vdash \Delta_2; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_{x:\text{act}}$.

We have

$$\begin{aligned}\textsf{share}\ x\ \{P \mid\mid Q\} &= \textsf{share}\ x\ \{(P_1 \mid\mid P_2) \mid\mid Q\} \\ &\equiv \underbrace{\textsf{share}\ x\ \{P_1 \mid\mid Q\}}_{R} \mid\mid P_2 \qquad (\equiv [\text{ShM}],\ x \in \text{fn}(P_1))\end{aligned}$$

By induction on $P_1 \downarrow_x$ and $Q \downarrow_x$ we conclude that $R \downarrow_{x:\text{act}}$.
Hence

$$\dfrac{\textsf{share}\ x\ \{P \mid\mid Q\} \equiv R \mid\mid P_2 \qquad \dfrac{R \downarrow_{x:\text{act}}}{(R \mid\mid P_2) \downarrow_{x:\text{act}}}\ [\text{mix}]}{(\textsf{share}\ x\ \{P \mid\mid Q\}) \downarrow_{x:\text{act}}}\ [\equiv]$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is [cut].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is [cut]. We have

$$\dfrac{P_1 \downarrow_{x:\text{act}} \qquad y \neq x}{P_1 \ |y|\ P_2 \downarrow_{x:\text{act}}}\ [\text{cut}]$$

where $P = P_1 \ |y|\ P_2$.
Since $P_1 \ |y|\ P_2 \vdash \Delta, x : \mathsf{U}_{\bullet} A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ and a type B for which $P_1 \vdash \Delta_1, y : \overline{B}, x : \mathsf{U}_{\bullet} A; \Gamma$ and $P_2 \vdash \Delta_2, y : B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_{x:\text{act}}$.
We have

$$\begin{aligned}\textsf{share}\ x\ \{P \mid\mid Q\} &= \textsf{share}\ x\ \{(P_1 \ |y|\ P_2) \mid\mid Q\} \\ &\equiv \underbrace{\textsf{share}\ x\ \{P_1 \mid\mid Q\}}_{R} \ |y|\ P_2 \qquad (\equiv [\text{CSh}],\ x, y \in \text{fn}(P_1))\end{aligned}$$

By induction on $P_1 \downarrow_{x:\text{act}}$ and $Q \downarrow_{x:\text{act}}$ we conclude that $(\textsf{share}\ x\ \{P_1 \mid\mid Q\}) \downarrow_{x:\text{act}}$.
Hence

$$\dfrac{\textsf{share}\ x\ \{P \mid\mid Q\} \equiv R \ |y|\ P_2 \qquad \dfrac{R \downarrow_{x} \qquad y \neq x}{(R \ |y|\ P_2) \downarrow_{x:\text{act}}}\ [\text{cut}]}{(\textsf{share}\ x\ \{P \mid\mid Q\}) \downarrow_{x:\text{act}}}\ [\equiv]$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is [cut!].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is [cut!]. We have

$$\dfrac{P_2 \downarrow_{x:\text{act}} \qquad z \neq x}{y.P_1 \ |!z : B|\ P_2 \downarrow_{x:\text{act}}}\ [\text{cut!}]$$

where $P = y.P_1 \ |!z : B|\ P_2$.
Since $y.P_1 \ |!z : B|\ P_2 \vdash \Delta, x : \mathsf{U}_{\bullet} A; \Gamma$ we conclude that $P_1 \vdash y : \overline{B}; \Gamma$ and $P_2 \vdash \Delta, x : \mathsf{U}_{\bullet} A; \Gamma, z : B$.

We have

$$\begin{aligned}\mathsf{share}\ x\ \{P \mid\mid Q\} &= \mathsf{share}\ x\ \{(y.P_1\ |!z:B|\ P_2) \mid\mid Q\} \\ &\equiv y.P_1\ |!z:B|\ \underbrace{(\mathsf{share}\ x\ \{P_2 \mid\mid Q\})}_{R} \quad (\equiv [\text{ShC!}]\ z \notin \text{fn}(Q))\end{aligned}$$

By induction on $P_2 \downarrow_{x:\text{act}}$ and $Q \downarrow_{x:\text{act}}$ we conclude that $R \downarrow_{x:\text{act}}$.
Hence

$$\dfrac{\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv y.P_1\ |!z|\ R \qquad \dfrac{R \downarrow_{x:\text{act}} \qquad z \neq x}{(y.P_1\ |!z|\ R) \downarrow_{x:\text{act}}}\ [\text{cut!}]}{(\mathsf{share}\ x\ \{P \mid\mid Q\}) \downarrow_{x:\text{act}}}\ [\equiv]$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is [share].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is [share]. We have

$$\dfrac{P_1 \downarrow_{x:\text{act}} \qquad y \neq x}{\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \downarrow_{x:\text{act}}}\ [\text{share}]$$

where $P = \mathsf{share}\ y\ \{P_1 \mid\mid P_2\}$.
The root rule of a derivation for $\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$ can be either [Tsh], [TshL] or [TshR]. We assume w.l.o.g. it is [Tsh]. The proof works in the same way for the other cases [TshL] and [TshR].
By inverting [Tsh] on $\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ, a type B for which $P_1 \vdash \Delta_1, y : \mathsf{U}_\bullet B, x : \mathsf{U}_\bullet A; \Gamma$ and $P_2 \vdash \Delta_2, y : \mathsf{U}_\bullet B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_{x:\text{act}}$.
We have

$$\begin{aligned}\mathsf{share}\ x\ \{P \mid\mid Q\} &= \mathsf{share}\ x\ \{\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \mid\mid Q\} \\ &\equiv \mathsf{share}\ y\ \{\underbrace{\mathsf{share}\ x\ \{P_1 \mid\mid Q\}}_{R} \mid\mid P_2\} \\ &\qquad (\equiv [\text{ShSh}],\ x, y \in \text{fn}(P_1))\end{aligned}$$

By induction on $P_1 \downarrow_{x:\text{act}}$ and $Q \downarrow_{x:\text{act}}$ we conclude that $(\mathsf{share}\ x\ \{P_1 \mid\mid Q\}) \downarrow_{x:\text{act}}$.
Hence

$$\dfrac{\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv \mathsf{share}\ y\ \{R \mid\mid P_2\} \qquad \dfrac{R \downarrow_{x:\text{act}} \qquad y \neq x}{(\mathsf{share}\ y\ \{R \mid\mid P_2\}) \downarrow_{x:\text{act}}}\ [\text{share}]}{(\mathsf{share}\ x\ \{P \mid\mid Q\}) \downarrow_{x:\text{act}}}\ [\equiv]$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is $[\leq]$.
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is $[\leq]$. We have

$$\dfrac{P \leq P' \qquad P' \downarrow_{x:\text{act}}}{P \downarrow_{x:\text{act}}}\ [\leq]$$

Since $P \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$, $P \leq P'$ and structural pre-congruence preserves typing, then $P' \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$.
By induction on $P' \downarrow_{x:\text{act}}, Q \downarrow_{x:\text{act}}$, we conclude that $\mathsf{share}\ x\ \{P' \mid\mid Q\} \downarrow_{x:\text{act}}$. Observe that

$$\mathsf{share}\ x\ \{P \mid\mid Q\}\ \leq \mathsf{share}\ x\ \{P' \mid\mid Q\} \qquad (\equiv [\text{cong2}])$$

Hence

$$\frac{\mathsf{share}\ x\ \{P \mid\mid Q\} \leq \mathsf{share}\ x\ \{P' \mid\mid Q\} \quad \mathsf{share}\ x\ \{P' \mid\mid Q\} \downarrow_{x:\text{act}}}{\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:\text{act}}}\ [\leq]$$

Lemma C.2(2) *Let* $P \vdash \Delta, x : \mathsf{U}_\circ A; \Gamma$, $Q \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$. *If* $P \downarrow_{x:act}$, *then* $\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:act}$.

Proof. By induction on the structure of a derivation for $P \downarrow_{x:act}$ and case analysis on the root rule. The base case [act] follows by applying $\leq$ rule [PSh] in order to expose the put action. For the inductive cases [mix], [cut], [cut!], [share] and] and [$\leq$] see the proof of Lemma C.2(1).

Case: The root rule of both $P \downarrow_{x:\text{act}}$ is [act]. We have

$$\frac{s(\mathcal{A}) = x}{\mathcal{A} \downarrow_{x:\text{act}}}\ [\text{act}]$$

where $P = \mathcal{A}$.
Since the subject of action $\mathcal{A}$- x - has the type $\mathsf{U}_\circ A$ (in the linear typing context), we conclude that $\mathcal{A}$ is a put action, i.e. $\mathcal{A} = \mathsf{put}\ x(y.P_1); P_2$ for some y, P_1, P_2.
By applying $\leq$ rule [PSh] we obtain

$$\mathsf{share}\ x\ \{\mathsf{put}\ x(y.P_1); P_2 \mid\mid Q\}\ \leq \mathsf{put}\ x(y.P_1); \underbrace{\mathsf{share}\ x\ \{P_2 \mid\mid Q\}}_{R} \quad (\leq [\text{PSh}])$$

Hence

$$\frac{\mathsf{share}\ x\ \{P \mid\mid Q\} \leq \mathsf{put}\ x(y.R); \quad \dfrac{s(\mathsf{put}\ x(y.P_1); R) = x}{\mathsf{put}\ x(y.P_1); R \downarrow_{x:\text{act}}}\ [\text{act}]}{\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:\text{act}}}\ [\leq]$$

Lemma C.2(3) *Let* $P \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$, $Q \vdash \Delta, x : \mathsf{U}_\circ A; \Gamma$. *If* $Q \downarrow_{x:act}$, *then* $\mathsf{share}\ x\ \{P \mid\mid Q\} \downarrow_{x:act}$.

Proof. Applying Lemma C.2(2) to $Q \vdash \Delta, x : \mathsf{U}_\circ A; \Gamma$ and $P \vdash \Delta, x : \mathsf{U}_\bullet A; \Gamma$ yields $\mathsf{share}\ x\ \{Q \mid\mid P\} \downarrow_{x:\text{act}}$.
By $\equiv$ rule [Sh] we have $\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv \mathsf{share}\ x\ \{Q \mid\mid P\}$.
Hence,

$$\frac{\mathsf{share}\ x\ \{P \mid\mid Q\} \equiv \mathsf{share}\ x\ \{Q \mid\mid P\} \quad \mathsf{share}\ x\ \{Q \mid\mid P\} x : \text{act}}{(\mathsf{share}\ x\ \{P \mid\mid Q\}) \downarrow_{x:\text{act}}}\ [\leq]$$

Lemma C.2(4) *Let* $P \vdash \Delta, x : \overline{A}; \Gamma$ *and* $Q \vdash \Delta', x : A; \Gamma$ *be processes for which* $P \downarrow_{x:act}$ *and* $Q \downarrow_{x:act}$*. Then,* $P \; |x| \; Q$ *reduces.*

Proof. By double induction on derivation trees for $P \downarrow_{x:\text{act}}$ and $Q \downarrow_{x:\text{act}}$. For the base cases we apply one of the principal cut reductions. For the inductive cases we consider that we are given a derivation tree for $P \downarrow_x$. This is w.l.o.g. since $P \; |x| \; Q \equiv Q \; |x| \; P$. For cases [mix], [cut], [cut!], [share] we commute the cut on x with the principal form of P by applying either $\equiv$ rule [CM], [CC], [CC!] or [CSh].The inductive case $P \downarrow_x$ rule [$\equiv$] follows immediately because the relation $\rightarrow$ is closed by structural congruence, i.e. satisfies $\rightarrow$ rule [$\equiv$].

Case: The root rule of both $P \downarrow_x$ and $Q \downarrow_x$ is [act]. We have

$$\frac{s(\mathcal{A}) = x}{\mathcal{A} \downarrow_x} \text{[act]} \quad \frac{s(\mathcal{B}) = x}{\mathcal{B} \downarrow_x} \text{[act]}$$

where $P = \mathcal{A}$ and $Q = \mathcal{B}$.
Since $\mathcal{A} \vdash \Delta, x : \overline{A}; \Gamma$ and $\mathcal{B} \vdash \Delta, x : A; \Gamma$ we conclude that $\mathcal{A}, \mathcal{B}$ is a pair of dual actions with the same subject. Hence, $P \; |x| \; Q$ reduces by applying one of the principal cut reductions.
For example, if $A = \bot$, we have

$$\mathcal{A} = \mathsf{close}\ x \quad \text{and} \quad \mathcal{B} = \mathsf{wait}\ x; Q'$$

Consequently

$$\mathsf{close}\ x \; |x| \; \mathsf{wait}\ x; Q' \rightarrow Q' \qquad (\rightarrow [\mathbf{1}\bot])$$

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is [mix].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is [mix]. We have

$$\frac{P_1 \downarrow_x}{(P_1 \; || \; P_2) \downarrow_x} \text{[mix]}$$

where $P = P_1 \; || \; P_2$.
Since $P_1 \; || \; P_2 \; \vdash \Delta, x : \overline{A}; \Gamma$ we conclude that there exists a partition Δ_1, Δ_2 of Δ s.t. $P_1 \vdash \Delta_1, x : \overline{A}; \Gamma$ and $P_2 \vdash \Delta_2; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_x$.
Then

$$\begin{aligned} P \; |x| \; Q &= (P_1 \; || \; P_2) \; |x| \; Q \\ &\equiv (P_1 \; |x| \; Q) \; || \; P_2 \qquad (\equiv [\text{CM}],\ x \in \text{fn}(P_1)) \end{aligned}$$

By induction on $P_1 \downarrow_x$ and $Q \downarrow_x$ we conclude that $P_1 \; |x| \; Q$, and hence $(P_1 \; |x| \; Q) \; || \; P_2$, reduces.

Case: Either the root rule of $P \downarrow_{x:\mathrm{act}}$ or the root rule of $Q \downarrow_{x:\mathrm{act}}$ is [cut].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\mathrm{act}}$ is [cut]. We have

$$\frac{P_1 \downarrow_x \quad y \neq x}{(P_1 \mid y \mid P_2) \downarrow_x}\ [\text{cut}]$$

where $P = P_1 \mid y \mid P_2$.
Since $P_1 \mid y \mid P_2 \ \vdash \Delta, x : \overline{A}; \Gamma$ we conclude that there exists a partition Δ_1, Δ_2 of Δ and a type B s.t. $P_1 \vdash \Delta_1, x : \overline{A}, y : \overline{B}; \Gamma$ and $P_2 \vdash \Delta_2, y : B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_x$.
Then

$$\begin{aligned} P \mid x \mid Q &= (P_1 \mid y \mid P_2) \mid x \mid Q \\ &\equiv (P_1 \mid x \mid Q) \mid y \mid P_2 && (\equiv [\text{CC}],\ x, y \in \text{fn}(P_1)) \end{aligned}$$

By induction on $P_1 \downarrow_x$ and $Q \downarrow_x$ we conclude that $P_1 \mid x \mid Q$, and hence $(P_1 \mid x \mid Q) \mid y \mid P_2$, reduces.

Case: Either the root rule of $P \downarrow_{x:\mathrm{act}}$ or the root rule of $Q \downarrow_{x:\mathrm{act}}$ is [cut!].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\mathrm{act}}$ is [cut!]. We have

$$\frac{P_2 \downarrow_x \quad z \neq x}{(y.P_1 \mid !z \mid P_2) \downarrow_x}\ [\text{cut!}]$$

where $P = y.P_1 \mid !z \mid P_2$.
Since $y.P_1 \mid !z \mid P_2 \ \vdash \Delta, x : \overline{A}; \Gamma$ we conclude that there exists a type B s.t. $P_1 \vdash y : \overline{B}; \Gamma$ and $P_2 \vdash \Delta, x : \overline{A}; \Gamma, z : B$.
Then

$$\begin{aligned} P \mid x \mid Q &= (y.P_1 \mid !z \mid P_2) \mid x \mid Q \\ &\equiv y.P_1 \mid !z \mid (P_2 \mid x \mid Q) && (\equiv [\text{CC!}],\ z \notin \text{fn}(Q)) \end{aligned}$$

By induction on $P_2 \downarrow_x$ and $Q \downarrow_x$ we conclude that $P_2 \mid x \mid Q$, and hence $y.P_1 \mid !z \mid (P_2 \mid x \mid Q)$, reduces.

Case: Either the root rule of $P \downarrow_{x:\mathrm{act}}$ or the root rule of $Q \downarrow_{x:\mathrm{act}}$ is [share].
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\mathrm{act}}$ is [share]. We have

$$\frac{P_1 \downarrow_x \quad y \neq x}{(\mathsf{share}\ y\ \{P_1 \parallel P_2\}) \downarrow_x}\ [\text{share}]$$

where $P = \mathsf{share}\ y\ \{P_1 \parallel P_2\}$.
The root rule of a derivation for $\mathsf{share}\ y\ \{P_1 \parallel P_2\} \vdash \Delta, x : \overline{A}; \Gamma$ can be either [Tsh], [TshL] or [TshR]. We assume w.l.o.g. it is [Tsh]. The proof works in the same way for the other cases [TshL] and [TshR].
By inverting [Tsh] on $\mathsf{share}\ y\ \{P_1 \parallel P_2\} \vdash \Delta, x : \overline{A}; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ, a type B for which $P_1 \vdash \Delta_1, y : \mathsf{U}_{\bullet} B, x : \overline{A}; \Gamma$ and

$P_2 \vdash \Delta_2, y : \mathsf{U}_{\bullet} B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_{x:\text{act}}$.
Then

$$\begin{aligned} P \;|x|\; Q \; &= (\mathsf{share}\ y\ \{P_1 \;||\; P_2\})\ |x|\ Q \\ &\equiv \mathsf{share}\ y\ \{(P_1\ |x|\ Q) \;||\; P_2\} \qquad (\equiv [\text{CSh}],\ x, y \in \text{fn}(P_1)) \end{aligned}$$

By induction on $P_1 \downarrow_x$ and $Q \downarrow_x$ we conclude that $P_1\ |x|\ Q$, and hence $\mathsf{share}\ y\ \{(P_1\ |x|\ Q) \;||\; P_2\}$, reduces.

Case: Either the root rule of $P \downarrow_{x:\text{act}}$ or the root rule of $Q \downarrow_{x:\text{act}}$ is $[\leq]$.
Suppose w.l.o.g. that the root rule of $P \downarrow_{x:\text{act}}$ is $[\leq]$. We have

$$\frac{P \leq P' \quad P' \downarrow_x}{P \downarrow_x}$$

Observe that since $P \vdash \Delta, x : \overline{A}; \Gamma$, $P \leq P'$ and structural pre-congruence preserves typing, then $P' \vdash \Delta, x : \overline{A}; \Gamma$.
By induction on $P' \downarrow_x$, $Q \downarrow_x$ we conclude that $P'\ |x|\ Q$ reduces. Since $P\ |x|\ Q \leq P'\ |x|\ Q$, $P\ |x|\ Q$ reduces as well (rule $\rightarrow [\leq]$).

Lemma C.2(5)) *Let $P \vdash \Delta, x : \overline{A}; \Gamma$, $Q \vdash \Delta', x : A; \Gamma$ be processes for which $P \downarrow_{x:fwd}$. Then, $P\ |x|\ Q$ reduces.*

Proof. By induction on a derivation trees for $P \downarrow_{x:\text{fwd}}$. We handle the base case, which follows by applying the principal cut conversion $\rightarrow$ [fwd]. For the inductive cases see the proof of Lemma C.2(4).

Case [fwd]
We have

$$\frac{}{\mathsf{fwd}\ x\ y \downarrow_x}\ [\text{fwd}]$$

where $P = \mathsf{fwd}\ x\ y$.
Then

$$\begin{aligned} \mathsf{fwd}\ x\ y\ |x|\ Q \; &\equiv \mathsf{fwd}\ y\ x\ |x|\ Q && (\equiv [\text{fwd}]) \\ &\rightarrow \{y/x\}Q && (\rightarrow [\text{fwd}]) \end{aligned}$$

Lemma C.2(6) *Let $P \vdash y : \overline{A}; \Gamma$ and $Q \vdash \Delta; \Gamma, x : A$ be processes for which $Q \downarrow_x$. Then, $y.P\ |!x|\ Q$ reduces.*

Proof. By induction on a derivation tree for $Q \downarrow_x$ and case analysis on the root rule. The base case [act] follows by applying the principal cut conversion $\rightarrow$ [call]. The inductive cases [mix], [cut], [cut!] and [share] follow by distributing the unrestricted cut over the arguments of Q (with $\equiv$ rules [D-C!M], [D-C!C], [D-C!C!] or[D-C!Sh]) and then apply the inductive hypothesis. The inductive case [$\equiv$] follows because reduction $\rightarrow$ is closed by structural congruence, i.e. satisfies rule $\rightarrow [\equiv]$.

Case: The root rule of $Q \downarrow_x$ is [act]. We have

$$\frac{s(\mathcal{A}) = x}{\mathcal{A} \downarrow_x}$$

where $Q = \mathcal{A}$.
Since $\mathcal{A} \vdash \Delta; \Gamma, x : A$, we have $\mathcal{A} = \mathsf{call}\ x(z); Q'$, for some Q'. Hence

$$y.P \ |!x|\ \mathsf{call}\ x(z); Q' \to \{z/y\}P \ |z|\ (y.P \ |!x|\ Q') \qquad (\to \text{[call]})$$

Case: The root rule of $Q \downarrow_x$ is [mix]. We have

$$\frac{Q_1 \downarrow_x}{(Q_1 \ ||\ Q_2) \downarrow_x}$$

where $Q = Q_1 \ ||\ Q_2$.
Since $Q_1 \ ||\ Q_2 \ \vdash \Delta; \Gamma, x : A$, there exists a partition Δ_1, Δ_2 of Δ for which $Q_1 \vdash \Delta_1; \Gamma, x : A$ and $Q_2 \vdash \Delta_2; \Gamma, x : A$.
We have

$$\begin{aligned} y.P \ |!x|\ Q &= y.P \ |!x|\ (Q_1 \ ||\ Q_2) \\ &\equiv (y.P \ |!x|\ Q_1) \ ||\ (y.P \ |!x|\ Q_2) \qquad (\equiv \text{[D-C!M]}) \end{aligned}$$

By induction on $Q_1 \downarrow_x$ we conclude that $y.P \ |!x|\ Q_1$, and hence $y.P \ |!x|\ Q$, reduces.

Case: The root rule of $Q \downarrow_x$ is [cut]. We have

$$\frac{Q_1 \downarrow_x \quad z \neq x}{(Q_1 \ |z|\ Q_2) \downarrow_x}$$

where $Q = Q_1 \ |z|\ Q_2$.
Since $Q_1 \ |z|\ Q_2 \ \vdash \Delta; \Gamma, x : A$, there exists a partition Δ_1, Δ_2 of Δ and a type B for which $Q_1 \vdash \Delta_1, z : \overline{B}; \Gamma, x : A$ and $Q_2 \vdash \Delta_2, z : B; \Gamma, x : A$.
We have

$$\begin{aligned} y.P \ |!x|\ Q &= y.P \ |!x|\ (Q_1 \ |z|\ Q_2) \\ &\equiv (y.P \ |!x|\ Q_1) \ |z|\ (y.P \ |!x|\ Q_2) \qquad (\equiv \text{[D-C!C]}) \end{aligned}$$

By induction on $Q_1 \downarrow_x$ we conclude that $y.P \ |!x|\ Q_1$, and hence $y.P \ |!x|\ Q$, reduces.

Case: The root rule of $Q \downarrow_x$ is [cut!]. We have

$$\frac{Q_2 \downarrow_x \quad z \neq x}{(w.Q_1 \ |!z|\ Q_2) \downarrow_x}$$

where $Q = w.Q_1 \ |!z|\ Q_2$.

Since $w.Q_1 \; |!z| \; Q_2 \; \vdash \Delta; \Gamma, x : A$, we conclude that exists a type B for which $Q_1 \vdash w : \overline{B}; \Gamma, x : A$ and $Q_2 \vdash \Delta; \Gamma, z : B, x : A$.
We have

$$\begin{aligned} y.P \; |!x| \; Q &= y.P \; |!x| \; (w.Q_1 \; |!z| \; Q_2) \\ &\equiv w.(y.P \; |!x| \; Q_1) \; |!z| \; (y.P \; |!x| \; Q_2) \qquad (\equiv \text{[D-C!C!]}) \end{aligned}$$

By induction on $Q_2 \downarrow_x$ we conclude that $y.P \; |!x| \; Q_2$, and hence $y.P \; |!x| \; Q$, reduces.

Case: The root rule of $Q \downarrow_x$ is [share]. We have

$$\frac{Q_1 \downarrow_x \quad z \neq x}{(\mathsf{share}\ z\ \{Q_1 \mid\mid Q_2\}) \downarrow_x}$$

where $Q = \mathsf{share}\ z\ \{Q_1 \mid\mid Q_2\}$.
Since $\mathsf{share}\ z\ \{Q_1 \mid\mid Q_2\} \; \vdash \Delta; \Gamma, x : A$, there are state flavours $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}$ and a partition $\Delta_1, \Delta_2, z : \mathbf{U}_{\mathcal{X}} \; B$ of Δ for which $Q_1 \vdash \Delta_1, z\mathbf{U}_{\mathcal{X}_1} \; B; \Gamma, x : A$,$Q_2 \vdash \Delta_2, z : \mathbf{U}_{\mathcal{X}_2} \; B; \Gamma, x : A$ and $\mathcal{X}_1 \oplus \mathcal{X}_2 = \mathcal{X}$.
The root rule of a derivation for $\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \vdash \Delta; \Gamma, x : A$ can be either [Tsh], [TshL] or [TshR]. We assume w.l.o.g. it is [Tsh]. The proof works in the same way for the other cases [TshL] and [TshR].
By inverting [Tsh] on $\mathsf{share}\ y\ \{P_1 \mid\mid P_2\} \vdash\vdash \Delta; \Gamma, x : A$ we conclude that exists a partition Δ_1, Δ_2 of Δ, a type B for which $P_1 \vdash \Delta_1, y : \mathsf{U}_{\bullet} B; \Gamma, x : A$ and $P_2 \vdash \Delta_2, y : \mathsf{U}_{\bullet} B; \Gamma, x : A$.
We have

$$\begin{aligned} y.P \; |!x| \; Q &= y.P \; |!x| \; (\mathsf{share}\ z\ \{Q_1 \mid\mid Q_2\}) \\ &\equiv \mathsf{share}\ z\ \{(y.P \; |!x| \; Q_1) \mid\mid (y.P \; |!x| \; Q_2)\} \qquad (\equiv \text{[D-C!Sh]}) \end{aligned}$$

By induction on $Q_1 \downarrow_x$ we conclude that $y.P \; |!x| \; Q_1$, and hence $y.P \; |!x| \; Q$ reduces.

Case: The root rule of $Q \downarrow_x$ is [$\leq$]. We have

$$\frac{Q \leq Q' \quad Q' \downarrow_x}{Q \downarrow_x}$$

Observe that since $Q \vdash \Delta; \Gamma, x : A$, $Q \leq Q'$ and structural pre-congruence preserves typing, we have $Q' \vdash \Delta; \Gamma, x : A$.
By induction on $Q' \downarrow_x$ we conclude that $y.P \; |!x| \; Q'$ reduces. Since $y.P \; |!x| \; Q \leq y.P \; |!x| \; Q'$, $y.P \; |!x| \; Q$ reduces as well ($\rightarrow$ rule [$\leq$]).

Lemma C.2(7) *Let $P \vdash \Delta, x : A; \Gamma$ and suppose that $A \neq \boldsymbol{S}_{\mathcal{X}} \; B$. If $P \downarrow_{x:fwd}$, then either (i) $P \downarrow_{y:fwd}$ for some $y : \overline{A} \in \Delta$ or (ii) P reduces.*

Proof. The proof is by structural induction on the derivation tree $P \downarrow_{x:\mathrm{fwd}}$ and case analysis on the root rule.

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is [fwd].

We have

$$\frac{}{\mathsf{fwd}\ x\ y \downarrow_{x:\text{fwd}}}\ [\text{fwd}]$$

where $P = \mathsf{fwd}\ x\ y$.
By inversion on $\mathsf{fwd}\ x\ y \vdash \Delta, x : A; \Gamma$ we conclude that $\Delta = y : \overline{A}$.
Observe that

$$\mathsf{fwd}\ x\ y \equiv \mathsf{fwd}\ y\ x \qquad (\equiv [\text{fwd}])$$

Then

$$\frac{\mathsf{fwd}\ x\ y \equiv \mathsf{fwd}\ y\ x \qquad \dfrac{}{\mathsf{fwd}\ y\ x \downarrow_{y:\text{fwd}}}\ [\text{fwd}]}{\mathsf{fwd}\ x\ y \downarrow_{y:\text{fwd}}}\ [\leq]$$

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is [mix].

We have

$$\frac{P_1 \downarrow_{x:\text{fwd}}}{(P_1 \mid\mid P_2) \downarrow_{x:\text{fwd}}}\ [\text{mix}]$$

where $P = P_1 \mid\mid P_2$.
By inversion on the typing judgment $P_1 \mid\mid P_2 \vdash \Delta, x : A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ s.t. $P_1 \vdash \Delta_1, x : A; \Gamma$ and $P_2 \vdash \Delta_2; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2 because $P_1 \downarrow_x$.
By induction on $P_1 \downarrow_{x:\text{fwd}}$, we conclude that either (i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1$ or (ii) P_1 reduces.

Case (i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1$.

Then

$$\frac{P_1 \downarrow_{y:\text{fwd}}}{(P_1 \mid\mid P_2) \downarrow_{y:\text{fwd}}}\ [\text{mix}]$$

Furthermore, since $y : \overline{A} \in \Delta_1$ and $\Delta = \Delta_1, \Delta_2$, then $y : \overline{A} \in \Delta$.

Case (ii) P_1 reduces.

Since reduction is a congruence, then $P_1 \mid\mid P_2$ reduces as well.

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is [cut].

We have

$$\frac{P_1 \downarrow_{x:\text{fwd}} \qquad z \neq x}{(P_1 \mid z \mid P_2) \downarrow_{x:\text{fwd}}}\ [\text{cut}]$$

where $P = P_1 \mid z \mid P_2$.
By inversion on the typing judgment $P_1 \mid z \mid P_2 \vdash \Delta, x : A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ and a type B s.t. $P_1 \vdash \Delta_1, x : A, z : \overline{B}; \Gamma$ and $P_2 \vdash \Delta_2, z : B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2 because $P_1 \downarrow_x$.
By induction on $P_1 \downarrow_{x:\text{fwd}}$, we conclude that either (i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1, z : \overline{B}$ or (ii) P_1 reduces. There are three cases to consider, depending on wether (i-i) $y \neq z$ or (i-ii) $y = z$.

Case (i-i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1$.
Then

$$\frac{P_1 \downarrow_{y:\text{fwd}} \quad y \neq z}{(P_1 \;|z|\; P_2) \downarrow_{y:\text{fwd}}} \;[\text{cut}]$$

Furthermore, since $y : \overline{A} \in \Delta_1$ and $\Delta = \Delta_1, \Delta_2$, then $y : \overline{A} \in \Delta$.

Case (i-ii) $P_1 \downarrow_{z:\text{fwd}}$ and $y = z$.
By Lemma C.2(5), we conclude that $P_1 \;|z|\; P_2$ reduces.

Case (ii) P_1 reduces.
Since reduction is a congruence, then $P_1 \;|z|\; P_2$ reduces as well.

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is [cut!].
We have

$$\frac{P_1 \downarrow_{x:\text{fwd}} \quad z \neq x}{(w.P_1 \;|!z|\; P_2) \downarrow_{x:\text{fwd}}} \;[\text{cut!}]$$

where $P = w.P_1 \;|!z|\; P_2$.
By inversion on the typing judgment $w.P_1 \;|!z|\; P_2 \vdash \Delta, x : A; \Gamma$ we conclude that exists a type B s.t. $P_1 \vdash w : \overline{B}; \Gamma$ and $P_2 \vdash \Delta, x : A; \Gamma, z : B$.
By induction on $P_2 \downarrow_{x:\text{fwd}}$, we conclude that either (i) $P_2 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta$ or (ii) P_2 reduces.

Case (i) $P_2 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta$. Then

$$\frac{P_2 \downarrow_{y:\text{fwd}} \quad y \neq z}{(w.P_1 \;|!z|\; P_2) \downarrow_{y:\text{fwd}}} \;[\text{cut!}]$$

Case (ii) P_2 reduces.
Since reduction is a congruence, then $w.P_1 \;|!z|\; P_2$ reduces as well.

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is [share].
We have

$$\frac{P_1 \downarrow_{x:\text{fwd}} \quad z \neq x}{(\mathsf{share}\; z\; \{P_1 \;||\; P_2\}) \downarrow_{x:\text{fwd}}} \;[\text{share}]$$

where $P = \mathsf{share}\; z\; \{P_1 \;||\; P_2\}$.
The root rule of a derivation for $\mathsf{share}\; y\; \{P_1 \;||\; P_2\} \vdash \Delta, x : A; \Gamma$ can be either [Tsh], [TshL] or [TshR]. We assume w.l.o.g. it is [Tsh]. The proof works in the same way for the other cases [TshL] and [TshR].
By inverting [Tsh] on $\mathsf{share}\; y\; \{P_1 \;||\; P_2\} \vdash \Delta, x : A; \Gamma$ we conclude that exists a partition Δ_1, Δ_2 of Δ, a type B for which $P_1 \vdash \Delta_1, z : \mathsf{U}_\bullet B, x : A; \Gamma$ and $P_2 \vdash \Delta_2, z : \mathsf{U}_\bullet B; \Gamma$. Observe that x lies in the linear typing context of P_1 and not of P_2, because $P_1 \downarrow_{x:\text{act}}$.
By induction on $P_1 \downarrow_{x:\text{fwd}}$, we conclude that either (i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1, z : \mathsf{U}_\bullet B$ or (ii) P_1 reduces.
Notice that, by hypothesis, $\overline{A} \neq \mathsf{U}_\bullet B$. Hence, $y : \overline{A} \in \Delta_1$.
There are then two cases to consider.

Case (i) $P_1 \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta_1$.

Then

$$\frac{P_1 \downarrow_{y:\text{fwd}} \quad y \neq z}{(\mathsf{share}\ z\ \{P_1 \mid\mid P_2\}) \downarrow_{y:\text{fwd}}}\ [\text{share}]$$

Case (ii) P_1 reduces

Since reduction is a congruence, then $\mathsf{share}\ z\ \{P_1 \mid\mid P_2\}$ reduces as well.

Case: The root rule of $P \downarrow_{x:\text{fwd}}$ is $[\leq]$.

We have

$$\frac{P \leq Q \quad Q \downarrow_{x:\text{fwd}}}{P \downarrow_{x:\text{fwd}}}\ [\leq]$$

Since $P \vdash \Delta, x : A; \Gamma$ and $P \leq Q$, then $Q \vdash \Delta, x : A; \Gamma$.
By induction on $Q \downarrow_{x:\text{fwd}}$ we conclude that either (i) $Q \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta$ or (ii) Q reduces.

Case (i) $Q \downarrow_{y:\text{fwd}}$ for some $y : \overline{A} \in \Delta$.

Then

$$\frac{P \leq Q \quad Q \downarrow_{y:\text{fwd}}}{P \downarrow_{y:\text{fwd}}}\ [\leq]$$

Case (ii) Q reduces.

Since reduction is closed by structural pre-congruence, then P reduces as well.

C.3 Liveness Lemma and Progress

We now state our liveness Lemma C.3 which says that a live open process either reduces or offers an interaction at some session x. This lemma implies our main progress result (Theorem C.1), with which we conclude this section.

Lemma C.3 (Liveness). *Let $P \vdash_\emptyset \Delta; \Gamma$ be a live process. Either $P \downarrow_x$, for some x, or P reduces.*

Proof. The proof is by structural induction on derivation tree for $P \vdash_\emptyset \Delta; \Gamma$ and case analysis on the root rule.

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is $[\text{T}\mathsf{0}]$.

We have

$$\frac{}{\mathsf{0} \vdash_\emptyset \emptyset; \Gamma}\ [\text{T}\mathsf{0}]$$

where $P = \mathsf{0}$. Holds vacuously because $\mathsf{0}$ is not live.

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [Tfwd].

We have

$$\frac{}{\mathsf{fwd}\ x\ y \vdash_\emptyset x : \overline{A}, y : A; \Gamma}\ [\text{Tfwd}]$$

Then

$$\frac{}{(\mathsf{fwd}\ x\ y) \downarrow_x}\ [\text{fwd}]$$

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [T**1**].

We have

$$\frac{}{\mathsf{close}\ x \vdash_\emptyset x : \mathbf{1}; \Gamma}\ [\text{T}\mathbf{1}]$$

where $P = \mathsf{close}\ x$. Observe that $\mathsf{close}\ x$ is an action. Then

$$\frac{s(\mathsf{close}\ x) = x}{\mathsf{close}\ x \downarrow_x}\ [\text{act}]$$

Similarly for the the other rules which introduce an action: [T⊥], [T⊗], [T⅋], [T$\oplus_l$], [T$\oplus_r$], [T&], [T?], [T!], [Tcall], [T∃], [T∀], [Tcorec], [Tμ], [Tν], [Taffine], [Tuse], [Tdiscard], [Tcell], [Tempty], [Trelease], [Ttake], [Tput].

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [Tvar].

We have

$$\frac{\eta = \eta', X(\vec{y}) \ \mapsto \Delta'; \Gamma'}{X(\vec{x}) \vdash_\emptyset \{\vec{x}/\vec{y}\}(\Delta'; \Gamma')}\ [\text{Tvar}]$$

where $P = X(\vec{x})$. Holds vacuously because assumes a nonempty η context.

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [Tmix].

We have

$$\frac{P_1 \vdash_\emptyset \Delta_1; \Gamma \quad P_2 \vdash_\emptyset \Delta_2; \Gamma}{P_1 \mid\mid P_2 \vdash_\emptyset \Delta_1, \Delta_2; \Gamma}\ [\text{Tmix}]$$

where $P = P_1 \mid\mid P_2$ and $\Delta = \Delta_1, \Delta_2$.
Since $P_1 \mid\mid P_2$ is live, then either P_1 is live or P_2 is live.
Suppose w.l.o.g. that P_1 is live. By induction on $P_1 \vdash_\emptyset \Delta_1; \Gamma$ we conclude that either $P_1 \downarrow_x$ or P_1 reduces.

Case $P_1 \downarrow_x$

Then

$$\frac{P_1 \downarrow_x}{(P_1 \mid\mid P_2) \downarrow_x}\ [\text{mix}]$$

Case P_1 reduces

Then, $P_1 \mid\mid P_2$ reduces because of $\rightarrow$ rule [cong].

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [Tcut].
We have

$$\frac{P_1 \vdash_\emptyset \Delta_1, x : \overline{A}; \Gamma \quad P_2 \vdash_\emptyset \Delta_2, x : A; \Gamma}{P_1 \;|x|\; P_2 \vdash_\emptyset \Delta_1, \Delta_2; \Gamma}\ [\text{cut}]$$

where $P = P_1 \;|x|\; P_2$ and $\Delta = \Delta_1, \Delta_2$.
Since both P_1 and P_2 have a nonempty linear typing context, we conclude that both P_1 and P_2 are live (lemma C.1).
By applying the i.h. to $P_1 \vdash_\emptyset \Delta_1, x : \overline{A}; \Gamma$ and $P_2 \vdash_\emptyset \Delta_2, x : A; \Gamma$ we conclude that

- $P_1 \downarrow_y$ or P_1 reduces, and
- $P_2 \downarrow_z$ or P_2 reduces

We have the following cases to consider
Case ($P_1 \downarrow_y$ and $y \neq x$) or ($P_2 \downarrow_z$ and $z \neq x$)
Suppose w.l.o.g. that $P_1 \downarrow_y$ and $y \neq x$.
Then

$$\frac{P_1 \downarrow_y \quad y \neq x}{(P_1 \;|x|\; P_2) \downarrow_y}\ [\text{cut}]$$

Case $P_1 \downarrow_x$ and $P_2 \downarrow_x$
We have the following two cases
Case $P_1 \downarrow_{x:\text{fwd}}$ or $P_2 \downarrow_{x:\text{fwd}}$
Suppose w.l.o.g. that $P_1 \downarrow_{x:\text{fwd}}$.
Then, by lemma C.2(3), we conclude that $P_1 \;|x|\; P_2$ reduces.
Case $P_1 \downarrow_{x:\text{act}}$ and $P_2 \downarrow_{x:\text{act}}$
Then, by lemma C.2(2), we conclude that $P_1 \;|x|\; P_2$ reduces.
Case P_1 reduces or P_2 reduces
Because of $\rightarrow$ rule [cong], $P_1 \;|x|\; P_2$ reduces.

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [Tcut!].
We have

$$\frac{P_1 \vdash_\emptyset y : \overline{B}; \Gamma \quad P_2 \vdash_\emptyset \Delta; \Gamma, x : A}{y.P_1 \;|!x|\; P_2 \vdash_\emptyset \Delta; \Gamma}\ [\text{cut!}]$$

where $P = y.P_1 \;|!x|\; P_2$.
Since $y.P_1 \;|!x|\; P_2$ is live, then P_2 is live.
By induction on $P_2 \vdash_\emptyset \Delta; \Gamma, x : A$ we conclude that either $P_2 \downarrow_z$ or P_2 reduces.
Case $P_2 \downarrow_z$ and $z \neq x$
Then

$$\frac{P_2 \downarrow_z \quad z \neq x}{(y.P_1 \;|!x|\; P_2) \downarrow_z}\ [\text{cut!}]$$

Case $P_2 \downarrow_x$
Then, $y.P_1 \;|!x|\; P_2$ reduces (lemma C.2(4)).

Case P_2 reduces

Because of $\to$ rule [cong], $y.P_1 \;|!x|\; P_2$ reduces.

Case: The root rule of $P \vdash_{\emptyset} \Delta; \Gamma$ is [Tsh].

We have

$$\frac{P_1 \vdash_{\emptyset} \Delta_1, x : \mathsf{U}_{\bullet} A; \Gamma \quad P_2 \vdash_{\emptyset} \Delta_2, x : \mathsf{U}_{\bullet} A; \Gamma}{\mathsf{share}\ x\ \{P_1 \mid\mid P_2\} \vdash_{\emptyset} \Delta_1, \Delta_2, x : \mathsf{U}_{\bullet} A; \Gamma}\ [\text{Tsh}]$$

where $P = \mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ and $\Delta = \Delta_1, \Delta_2, x : \mathsf{U}_{\bullet} A$.
Since both P_1 and P_2 type with a nonempty linear context and an empty η, then both P_1 and P_2 are live (Lemma C.1).
By applying the i.h. to $P_1 \vdash_{\emptyset} \Delta_1, x : \mathsf{U}_{\bullet} A; \Gamma$ and $P_2 \vdash_{\emptyset} \Delta_2, x : \mathsf{U}_{\bullet} A; \Gamma$ we conclude both

- $P_1 \downarrow_y$ or P_1 reduces, and
- $P_2 \downarrow_z$ or P_2 reduces

We have the following cases to consider.

Case A ($P_1 \downarrow_y$ and $y \neq x$) or ($P_2 \downarrow_z$ and $z \neq x$)

Suppose w.l.o.g. that $P_1 \downarrow_y$ and $y \neq x$.
Then

$$\frac{P_1 \downarrow_y \quad y \neq x}{(\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}) \downarrow_y}\ [\text{share}]$$

Case B $P_1 \downarrow_x$ and $P_2 \downarrow_x$

We have the following two cases.

Case B1 $P_1 \downarrow_{x:\text{fwd}}$ or $P_2 \downarrow_{x:\text{fwd}}$

Suppose w.l.o.g. that $P_1 \downarrow_{x:\text{fwd}}$.
Observe that x occurs typed by $\mathsf{U}_{\bullet} A$ in the linear typing context of P_1. Hence, we can apply Lemma C.2(7) in order to conclude that either (i) $P_1 \downarrow_y$ for $y \neq x$ or (ii) P_1 reduces. If (i) go to case A. If (ii), go to case C.

Case B2 $P_1 \downarrow_{x:\text{act}}$ and $P_2 \downarrow_{x:\text{act}}$.

Then $(\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}) \downarrow_x$ (Lemma C.2(1)).

Case C P_1 reduces or P_2 reduces

Because of $\to$ rule [cong], $\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ reduces.

Case: The root rule of $P \vdash_{\emptyset} \Delta; \Gamma$ is [TshL].

We have

$$\frac{P_1 \vdash_{\emptyset} \Delta_1, x : \mathsf{U}_{\circ} A; \Gamma \quad P_2 \vdash_{\emptyset} \Delta_2, x : \mathsf{U}_{\bullet} A; \Gamma}{\mathsf{share}\ x\ \{P_1 \mid\mid P_2\} \vdash_{\emptyset} \Delta_1, \Delta_2, x : \mathsf{U}_{\circ} A; \Gamma}\ [\text{TshL}]$$

where $P = \mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ and $\Delta = \Delta_1, \Delta_2, x : \mathsf{U}_{\circ} A$.
By applying the i.h. to $P_1 \vdash_{\emptyset} \Delta_1, x : \mathsf{U}_{\circ} A; \Gamma$ we conclude that either $P_1 \downarrow_y$ or P_1 reduces.
We have the following cases to consider.

Case A $P_1 \downarrow_y$ and $y \neq x$
Then

$$\frac{P_1 \downarrow_y \quad y \neq x}{(\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}) \downarrow_y}\ [\text{share}]$$

Case B $P_1 \downarrow_x$
We have the following two cases.

Case B1 $P_1 \downarrow_{x:\text{fwd}}$
Suppose w.l.o.g. that $P_1 \downarrow_{x:\text{fwd}}$.
Observe that x occurs typed by $\mathsf{U}_\circ A$ in the linear typing context of P_1. Hence, we can apply Lemma C.2(7) in order to conclude that either (i) $P_1 \downarrow_y$ for $y \neq x$ or (ii) P_1 reduces. If (i) go to case A. If (ii), go to case C.

Case B2 $P_1 \downarrow_{x:\text{act}}$.
Then $(\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}) \downarrow_x$ (Lemma C.2(2)).

Case C P_1 reduces or P_2 reduces
Because of $\rightarrow$ rule [cong], $\mathsf{share}\ x\ \{P_1 \mid\mid P_2\}$ reduces.

Case: The root rule of $P \vdash_\emptyset \Delta; \Gamma$ is [TshR].
Similar to case [TshL].

Theorem C.1 (Progress). *Let $P \vdash_\emptyset \emptyset; \emptyset$ be a live process. Then, P reduces.*

Proof. Follows from Lemma C.3 since $\mathsf{fn}(P) = \emptyset$.

D Strong Normalisation

We prove that reduction $\rightarrow$ satisfies strong normalisation (Theorem 3.3). First, we equip the operational model $\rightarrow$ with interference-sensitive cells, they allow us to reason about state interference compositionally (Subsection D.1). Next, we introduce the logical predicates $[\![x : A]\!]_\sigma$ for strong normalisation (Subsection D.4). Finally, we prove the Fundamental Lemma D.11, from which SN follows. In this section, we work with binary relation $\approx$, that includes structural pre-congruence $\leq$, but adds a complete set of commuting conversions, along standard lines [21, 26, 74, 61], which allows to commute actions with the static constructs mix, cut and share, for example:

$$\begin{array}{ll}(\mathsf{wait}\ x; P)\ |y|\ Q & \approx \mathsf{wait}\ x; (P\ |y|\ Q),\ y \neq x \\ \mathsf{share}\ y\ \{\mathsf{wait}\ x; P \mid\mid Q\} & \approx \mathsf{wait}\ x; \mathsf{share}\ y\ \{P \mid\mid Q\}\end{array}$$

Relation $\approx$ essentially plays the role of the labelled transition system in the proof of strong normalisation given in [58].

D.1 Interference-Sensitive Reference Cells

We equip the operational model $\to$ with interference-sensitive cells, reference cells which internalise state interference, resultant from shared usage manipulation, in their operational model. These auxiliary process constructs play a crucial technical role in the proof of the strong normalisation result, essentially because they allow us to reason about state interference compositionally, as expressed by Lemma D.4. We start with the definition of interference-sensitive cells.

Definition D.1 (Interference-Sensitive Reference Cells). *Let $S \subseteq \{R \mid R \vdash_\eta a : \wedge A\}$. We extend the process calculus* CLASS *with the interference-sensitive full* $\mathsf{cell}\ c(a.S)$ *and empty* $\mathsf{empty}\ c(a.S)$ *cells, which have following associated principal reduction rules*

$$\begin{array}{lll}
\mathsf{cell}\ c(a.S)\ |c|\ \mathsf{release}\ c & \to P\ |a|\ \mathsf{discard}\ a,\ P \in S & (1)\\
\mathsf{cell}\ c(a.S)\ |c|\ \mathsf{take}\ c(a'); Q & \to \mathsf{empty}\ c(a.S)\ |c|\ (P\ |a|\ \{a/a'\}Q),\ P \in S & (2)\\
\mathsf{empty}\ c(a.S)\ |c|\ \mathsf{put}\ c(a.Q_1); Q_2 & \to \mathsf{cell}\ c(a.S)\ |c|\ Q_2 & (3)
\end{array}$$

Rules (1) and (2) apply to usage processes $P \vdash c : \mathsf{U}_\bullet A$, whereas rule (3) applies to a usage process $P \vdash c : \mathsf{U}_\circ A$. When a take or a release action interacts with an interference-sensitive full cell $\mathsf{cell}\ c(a.S)$ we pick an arbitrary element P from the set S (rules (1) and (2)). On the other hand, when a put action $\mathsf{put}\ c(a.Q_1); Q_2$ interacts with an interference-sensitive empty cell $\mathsf{empty}\ c(a.S)$ it evolves to $\mathsf{cell}\ c(a.S)$ (3).

The process constructs $\mathsf{cell}\ c(a.S)$ and $\mathsf{empty}\ c(a.S)$ can be though of as reference cells subject to interference over the set S. They contrast with the the basic empty and full reference cells $\mathsf{cell}\ c(a.P)$ and $\mathsf{empty}\ c$ of CLASS which are, so to speak, blind to the interference that results from concurrency, since from a local point of view they obey a sequential protocol: if a cell is not being shared by any other thread then every take acquires the session that was put before or that was present in the cell initially. On the other hand, a take on an interference-sensitive cell might obtain a session distinct from the session previously put, even if the interference-sensitive cell is not being explicitly shared. So, interference resulting from cell sharing is baked in the operational semantics of the interference-sensitive cells as expressed by rules (1)-(3) of Def. D.1. Provided the usages are well-behaved according to to the set over which the interference-sensitive cells are defined, as formalised by coinductive Def. D.2, it is possible to simulate the basic full and empty cells of CLASS with interference-sensitive cells, as described by Lemma D.2.

Definition D.2. *Let $S \subseteq \{R \mid R \vdash y : \wedge \overline{A}\}$. A process P, where either $P \vdash x : \mathsf{U}_\bullet A$ or $P \vdash x : \mathsf{U}_\circ A$, is S-preserving on x iff the following hold*

(a) *If $P \xrightarrow{*} Q$, $Q \approx \mathsf{take}\ x(y'); Q'$ and $R \in S$, then $\{y'/y\}R\ |y'|\ Q'$ is S-preserving on x.*

(b) *If $P \xrightarrow{*} Q$ and $Q \approx \mathsf{put}\ x(y'.Q_1); Q_2$, then $\{y/y'\}Q_1 \in S$ and Q_2 is S-preserving on x.*

If a process P is S-preserving on x and after some internal reductions it offers a take action, then the continuation of the take action composed with an element from S is also S-preserving on x (Def. D.2(a)). Dually, if P offers a put action then the element put is on the set S and the continuation is still S-preserving (Def. D.2(b)). The notion of S-preserving is preserved by reduction $\xrightarrow{*}$, as expressed by the following lemma.

Lemma D.1. *If P is S-preserving on x and $P \xrightarrow{*} Q$, then Q is S-preserving on x.*

Proof. Immediate from Def. D.2.

The following result sufficient conditions for simulating be basic reference cells using the interference-sensitive cells. But before we need to introduce the notion of simulation. A simulation $\mathcal{S}$ is a binary relation on processes s.t. whenever $(P, Q) \in \mathcal{S}$ and $P \to P'$ then there exists Q' s.t. $Q \xrightarrow{+}_{\mathsf{c}} Q'$ and $(P', Q') \in \mathcal{S}$. We say that P simulates Q iff there exists a simulation $\mathcal{S}$ s.t. $(Q, P) \in \mathcal{S}$.

Lemma D.2. *The following properties hold*

(1) *Let $S \subseteq \{R \mid R \vdash_\eta y : \wedge A\}$, $P \in S$, $Q \vdash_\eta x : \mathsf{U}_\bullet \overline{A}$ and suppose Q is S-preserving on x. Then, $\mathsf{cell}\ x(y.P) \ |x|\ Q$ is simulated by $\mathsf{cell}\ x(y.S) \ |x|\ Q$.*
(2) *Let $S \subseteq \{R \mid R \vdash_\eta y : \wedge A\}$, $Q \vdash_\eta x : \mathsf{U}_\circ \overline{A}$ and Q suppose Q is S-preserving on x. Then, $\mathsf{empty}\ x \ |x|\ Q$ is simulated by $\mathsf{empty}\ x(y.S) \ |x|\ Q$.*

Proof. Define

$$\mathcal{S} \triangleq \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$$

where

$$\begin{aligned}
\mathcal{S}_1 \triangleq \{(M,N) \mid\ & \exists P \in S, \exists Q \vdash_\eta x : \mathsf{U}_\bullet \overline{A}.\ Q \text{ is } S\text{-preserving on } x \text{ and} \\
& M \approx \mathsf{cell}\ x(y.P) \ |x|\ Q \text{ and } N \approx \mathsf{cell}\ x(y.S) \ |x|\ Q\} \\
\mathcal{S}_2 \triangleq \{(M,N) \mid\ & \exists Q \vdash_\eta x : \mathsf{U}_\bullet \overline{A}.\ Q \text{ is } S\text{-preserving on } x \text{ and} \\
& M \approx \mathsf{empty}\ x \ |x|\ Q \ \text{ and } N \approx \mathsf{empty}\ x(y.S) \ |x|\ Q\} \\
\mathcal{S}_3 \triangleq \{(M,N) \mid\ & M \approx N\}
\end{aligned}$$

We prove that $\mathcal{S}$ is a simulation. Suppose $(M, N) \in \mathcal{S}$ and $M \to M'$. We perform first case analysis on $(M, N) \in \mathcal{S}$.

Case: $(M, N) \in \mathcal{S}_1$. Then

$$M \approx \mathsf{cell}\ x(y.P) \ |x|\ Q$$

and

$$N \approx \mathsf{cell}\ x(y.S) \ |x|\ Q$$

where $P \in S$ and $Q \vdash_\eta x : \mathsf{U}_\bullet \overline{A}$.
We perform case analysis on the reduction $M \to M'$.

Case: Internal reduction of Q.

Then
$$M' \approx \mathsf{cell}\ x(y.P)\ |x|\ Q'$$
Let
$$N' \triangleq \mathsf{cell}\ x(y.S)\ |x|\ Q'$$
Then, $N \to N'$ and $(M', N') \in \mathcal{S}_1$.

Case: Cell-take interaction on session x.

Then, $Q \approx \mathsf{take}\ x(y); Q'$ and
$$M' \approx \mathsf{empty}\ x\ |x|\ (R\ |y|\ Q')$$
where $R \in S$.
Since, by hypothesis, $Q \vdash_\eta x : \mathsf{U}_\bullet\overline{A}$ and $Q \approx \mathsf{take}\ x(y); P'$, then $Q' \vdash_\eta x : \mathsf{U}_\circ\overline{A}, y : \vee\overline{A}$. Since $R \in S$, then $R \vdash_\eta y : \wedge A$, hence $R\ |y|\ P' \vdash_\eta x : \mathsf{U}_\circ\overline{A}$ is S-preserving (Def. D.2(a)).
Let
$$N' \triangleq \mathsf{empty}\ x(y.S)\ |x|\ (R\ |y|\ Q')$$
Then, $N \to N'$ and $(M', N') \in \mathcal{S}_2$.

Case: Cell-release interaction on session x.

Then, $Q \approx \mathcal{C}[\mathsf{release}\ x]$ and
$$\begin{aligned} M &\approx \mathsf{cell}\ x(y.P)\ |x|\ \mathcal{C}[\mathsf{release}\ x] \\ &\to \mathcal{C}[P\ |y|\ \mathsf{discard}\ y] \end{aligned}$$
Let
$$N' \triangleq \mathcal{C}[P\ |y|\ \mathsf{discard}\ y]$$
Then, since $P \in S$:
$$\begin{aligned} N &\approx \mathsf{cell}\ x(y.S)\ |x|\ \mathcal{C}[\mathsf{release}\ x] \\ &\to \mathcal{C}[P\ |y|\ \mathsf{discard}\ y] = N' \end{aligned}$$
and $(M', N') \in \mathcal{S}_3$.

Case: $(M, N) \in \mathcal{S}_2$. Then
$$M \approx \mathsf{empty}\ x\ |x|\ Q$$
and
$$N \approx \mathsf{empty}\ x(y.S)\ |x|\ Q$$
where $Q \vdash_\eta x : \mathsf{U}_\bullet\overline{A}$.
We perform case analysis on the reduction $M \to M'$.

Case: Internal reduction of Q.

Then
$$M' \approx \mathsf{empty}\ x\ |x|\ Q'$$
Let
$$N' \triangleq \mathsf{empty}\ x(y.S)\ |x|\ Q'$$
Then, $N \to N'$ and $(M', N') \in \mathcal{S}_2$.

Case: Cell-put interaction on session x.
Then, $Q \approx \mathsf{put}\ x(y.Q_1); Q_2$.
By hypothesis, $Q \vdash_\eta x : \mathsf{U}_\circ \overline{A}$, hence $Q_2 \vdash_\eta x : \mathsf{U}_\bullet \overline{A}$.
Furthermore, since Q is S-preserving on x, then $Q_1 \in S$ and Q_2 is S-preserving on x (Def. D.2(b)).
Then

$$M' \approx \mathsf{cell}\ x(y.Q_1)\ |x|\ Q_2$$

Let

$$N' \triangleq \mathsf{cell}\ x(y.S)\ |x|\ Q_2$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_1$.

Case: $(M, N) \in \mathcal{S}_3$.
Trivial since $M \approx N$.

Crucially, the notion of S-preserving is preserved by concurrent share composition as described by the following lemma

Lemma D.3. *If P and Q are S-preserving on x, then $\mathsf{share}\ x\ \{P \mid\mid Q\}$ is S-preserving on x.*

Proof. By coinduction. We need to prove that $\mathsf{share}\ x\ \{P \mid\mid Q\}$ satisfies (a)-(b) of Def. D.2.

(a) Let $R \in S$ and suppose $\mathsf{share}\ x\ \{P \mid\mid Q\} \xrightarrow{*} \mathsf{take}\ x(y); M$.
The take on x comes either from P or Q. Suppose w.l.o.g. that it comes from P. Then

$$P \xrightarrow{*} \mathsf{take}\ x(y); P' \text{ and } M \approx \mathsf{share}\ x\ \{P' \mid\mid Q'\}$$

where $Q \xrightarrow{*} Q'$.
We need to prove that $R\ |y|\ M$ is S-preserving on x.
But

$$R\ |y|\ M \approx R\ |y|\ \mathsf{share}\ x\ \{P' \mid\mid Q'\} \approx \mathsf{share}\ x\ \{R\ |y|\ P' \mid\mid Q'\}$$

Since P is S-preserving on x and $R \in S$, then Def. D.2(a) implies that $R\ |y|\ P'$ is S-preserving on x.
Since Q is S-preserving on x and $Q \xrightarrow{*} Q'$, then Q' is S-preserving on x (by Lemma D.1).
By coinductive hypothesis we conclude that $\mathsf{share}\ x\ \{R\ |y|\ P' \mid\mid Q'\}$ is S-preserving on x.

(b) If $P \xrightarrow{*} Q$ and $Q \approx \mathsf{put}\ x(y.Q_1); Q_2$, then $Q_1 \in S$ and Q_2 is S-preserving on x.
Suppose $\mathsf{share}\ x\ \{P \mid\mid Q\} \xrightarrow{*} \mathsf{put}\ x(y.M_1); M_2$.
Suppose w.l.o.g. that $P \vdash x : \mathsf{U}_\circ A$, then the put comes from P.
Hence

$$P \xrightarrow{*}\approx \mathsf{put}\ x(y.M_1); P' \text{ and } M \approx \mathsf{share}\ x\ \{P' \mid\mid Q'\}$$

where $Q \xrightarrow{*} Q'$.

We need to prove that (i) $M_1 \in S$ and that (ii) $\mathsf{share}\ x\ \{P' \mid\mid Q'\}$ is S-preserving on x.
(i) follows since P is S-preserving on x (Def. D.2(b)).
Since P is S-preserving on x (Def. D.2(b)), then P' is S-preserving.
Since Q is S-preserving on x and $Q \xrightarrow{*} Q'$, then Q' is S-preserving on x (by Lemma D.1).
By coinductive hypothesis, $\mathsf{share}\ x\ \{P' \mid\mid Q'\}$ is S-preserving on x, hence (ii).

Since the potential interference resulting from cell sharing is absorbed by the operational semantics that characterises the interference-sensitive cells (Def. D.1), we have the following simulation property which allows us to reason modularly about state sharing, and with which we conclude this section.

Lemma D.4. *The following pair of simulations hold*

(1) *Let* $P \vdash_\eta x : \mathsf{U}_\bullet A$, $Q \vdash_\eta x : \mathsf{U}_\bullet A$ *and* $S \subseteq \{R \mid R \vdash_\eta y : \wedge\overline{A}\}$. *Then,*

$$(\mathsf{cell}\ x(y.S)\ |x|\ P) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$
simulates
$$\mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\}$$

(2) *Let* $P \vdash_\eta x : \mathsf{U}_\circ A$, $Q \vdash_\eta x : \mathsf{U}_\bullet A$ *and* $S \subseteq \{R \mid R \vdash_\eta y : \wedge\overline{A}\}$. *Then,*

$$(\mathsf{empty}\ x(y.S)\ |x|\ P) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$
simulates
$$\mathsf{empty}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\}$$

Proof. Define
$$\mathcal{S} \triangleq \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$$
where

$$\begin{aligned}
\mathcal{S}_1 \triangleq \{(M,N) \mid\ & \exists P \vdash_\eta x : \mathsf{U}_\bullet A, \exists Q \vdash_\eta x : \mathsf{U}_\bullet A.\ M \approx \mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\} \\
& \text{and } N \approx (\mathsf{cell}\ x(y.S)\ |x|\ P) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)\} \\
\mathcal{S}_2 \triangleq \{(M,N) \mid\ & \exists P \vdash_\eta x : \mathsf{U}_\circ A, \exists Q \vdash_\eta x : \mathsf{U}_\bullet A.\ M \approx \mathsf{empty}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\} \\
& \text{and } N \approx (\mathsf{empty}\ x(y.S)\ |x|\ P) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)\} \\
\mathcal{S}_3 \triangleq \{(M,N) \mid\ & \exists P \vdash_\eta \emptyset;\emptyset, \exists\mathcal{C}\exists\mathcal{D}.\ M \approx \mathcal{C} \circ \mathcal{D}[P] \ \text{ and } N \approx \mathcal{C}[P] \mid\mid \mathcal{D}[P]\}
\end{aligned}$$

We prove that $\mathcal{S}$ is a simulation. Suppose $(M,N) \in \mathcal{S}$ and $M \to M'$. We perform first case analysis on $(M,N) \in \mathcal{S}$.

Case: $(M,N) \in \mathcal{S}_1$. Then

$$M \approx \mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\}$$

and
$$N \approx (\mathsf{cell}\ x(y.S)\ |x|\ P) \mid\mid (S\ |x|\ Q)$$

where $P \vdash_\eta x : \mathsf{U}_\bullet A$ and $Q \vdash_\eta x : \mathsf{U}_\bullet A$.
We perform case analysis on the reduction $M \to M'$.

Case: Internal reduction of either P or Q.

Suppose w.l.o.g. that $M \to M'$ is obtained by an internal reduction $P \to P'$.

Then

$$M' \approx \mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P' \mid\mid Q\}$$

Let

$$N' \triangleq (\mathsf{cell}\ x(y.S)\ |x|\ P') \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_1$.

Case: Cell-take interaction on session x.

Suppose w.l.o.g. that the interaction occurs between the cell and P.

Then, $P \approx \mathsf{take}\ x(y); P'$ and

$$M' \approx \mathsf{empty}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{R\ |y|\ P' \mid\mid Q\}$$

where $R \in S$.

Since, by hypothesis, $P \vdash_\eta x : \mathsf{U}_\bullet A$ and $P \approx \mathsf{take}\ x(y); P'$, then $P' \vdash_\eta x : \mathsf{U}_\circ A, y : \vee A$. Since $R \in S$, then $R \vdash_\eta y : \wedge\overline{A}$, hence $R\ |y|\ P' \vdash_\eta x : \mathsf{U}_\circ A$.

Let

$$N' \triangleq (\mathsf{empty}\ x(y.S)\ |x|\ (R\ |y|\ P')) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_2$.

Case: Cell-release interaction on session x.

Both $P \approx \mathcal{C}[\mathsf{release}\ x]$ and $Q \approx \mathcal{D}[\mathsf{release}\ x]$, for some static contexts $\mathcal{C}, \mathcal{D}$.

Then

$$\begin{aligned} M &\approx \mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{\mathcal{C}[\mathsf{release}\ x] \mid\mid \mathcal{D}[\mathsf{release}\ x]\} \\ &\approx \mathcal{C} \circ \mathcal{D}[\mathsf{cell}\ x(y.S)\ |x|\ \mathsf{release}\ x] \\ &\to \mathcal{C} \circ \mathcal{D}[R\ |y|\ \mathsf{discard}\ y] \end{aligned}$$

where $R \in S$.

Let

$$N' \triangleq \mathcal{C}[R\ |y|\ \mathsf{discard}\ y] \mid\mid \mathcal{D}[R\ |y|\ \mathsf{discard}\ y]$$

Then $N \xrightarrow{2}_{\mathsf{c}} N'$ and $(M', N') \in \mathcal{S}_3$.

Case: $(M, N) \in \mathcal{S}_2$. Then

$$M \approx \mathsf{empty}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P \mid\mid Q\}$$

and

$$N \approx (\mathsf{empty}\ x(y.S)\ |x|\ P) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$

where $P \vdash_\eta x : \mathsf{U}_\circ A$ and $Q \vdash_\eta x : \mathsf{U}_\bullet A$.

We perform case analysis on the reduction $M \to M'$.

Case: Internal reduction of either P or Q.

Suppose w.l.o.g. that $M \to M'$ is obtained by an internal reduction $P \to P'$.

Then

$$M' \approx \mathsf{empty}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P' \mid\mid Q\}$$

Let

$$N' \triangleq (\mathsf{empty}\ x(y.S)\ |x|\ P') \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_2$.

Case: Cell-put interaction on session x.

Then, $P \approx \mathsf{put}\ x(y.P_1); P_2$.

By hypothesis, $P \vdash_\eta x : \mathsf{U}_\circ A$, hence $P_2 \vdash_\eta x : \mathsf{U}_\bullet A$.

Then

$$M' \approx \mathsf{cell}\ x(y.S)\ |x|\ \mathsf{share}\ x\ \{P_2 \mid\mid Q\}$$

Let

$$N' \triangleq (\mathsf{cell}\ x(y.S)\ |x|\ P_2) \mid\mid (\mathsf{cell}\ x(y.S)\ |x|\ Q)$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_1$.

Case: $(M, N) \in \mathcal{S}_3$.

Then

$$M \approx \mathcal{C} \circ \mathcal{D}[P]$$

and

$$N \approx \mathcal{C}[P] \mid\mid \mathcal{D}[P]$$

where $P \vdash_\eta \emptyset; \emptyset$.

We perform case analysis on the reduction $M \to M'$.

Case: Internal reduction of either $\mathcal{C}$ or $\mathcal{D}$.

Suppose w.l.o.g. that $\mathcal{C} \to \mathcal{C}'$. Then

$$M' \approx \mathcal{C}' \circ \mathcal{D}[P]$$

Let

$$N' \triangleq \mathcal{C}'[P] \mid\mid \mathcal{D}[P]$$

Then, $N \to N'$ and $(M', N') \in \mathcal{S}_3$.

Case: Internal reduction of P.

Suppose $P \to P'$.

Then

$$M \approx \mathcal{C} \circ \mathcal{D}[P']$$

Let $N' \triangleq \mathcal{C}[P'] \mid\mid \mathcal{D}[P']$.

Then, $N \xrightarrow{2}_{\mathsf{c}} N'$ and $(M', N') \in \mathcal{S}_3$.

D.2 Logical Predicates $[\![x : A]\!]_\sigma$.

The goal of this section is to introduce the linear logical predicates, used to establish our strong normalisation result. In D.3, we start by presenting some basic properties about SN processes and then we introduce the orthogonal operation. This operation is then used to define, later in D.4, our basic logical predicates $[\![x : A]\!]_\sigma$, we then prove some properties. We conclude in D.5 with the proof of the Fundamental Lemma D.11, from which our strong normalisation result follows immediately (Theorem 3.3).

D.3 Orthogonal and Basic Properties

We start by stating some basic properties (Lemma D.5) but first let us introduce a measure on SN processes, which will be often used to prove properties about strong normalisation by induction. For every process P there is a finite (up to $\approx$) number of processes Q for which $P \to Q$. Hence, By König's Lemma, for each SN process P there is a longest $\to$-reduction sequence starting with P, we denote the length of this sequence by $N(P)$.

Lemma D.5 (SN: Basic Properties). *The following properties hold*

(1) *If P is SN and $P \approx Q$, then Q is SN.*
(2) *If P is SN and $P \to Q$, then Q is SN.*
(3) *Suppose Q is SN whenever $P \to Q$. Then, P is SN.*
(4) *If P and Q are SN, then $P \mid\mid Q$ is SN.*
(5) *If Q is SN and Q simulates P, then P is SN.*

Proof. All properties are easy to establish, in particular we have the following: in (1) $N(P) = N(Q)$, in (2) $N(Q) = N(P) - 1$, in (3) $N(P) = (\mathsf{max}\ \{Q \mid P \to Q\}) + 1$ and in (4) $N(P \mid\mid Q) = N(P) + N(Q)$.

We will now introduce the orthogonal, which will play a key role when defining logical predicates for strong normalisation. As we will see, each logical predicate is defined by taking the orthogonal of some set. In the following, we write P_x to emphasise that x is the only free name of P.

Definition D.3 (Orthogonal $(-)^\perp$). *Let S be a subset of processes Q_x with a single free name x. We define the orthogonal of S, written $S^\perp$, by*

$$S^\perp \triangleq \{P_x \mid \forall Q_x \in S.\ P_x\ |x|\ Q_x \textit{ is SN}\}$$

The orthogonal satisfies some well-known properties, as stated by the following lemma.

Lemma D.6 (Orthogonal: Basic Properties). *The following properties hold*

(1) *If $P \in S^\perp$ and $P \approx Q$, then $Q \in S^\perp$.*

(2) *If $P \in S^\perp$ and $P \to Q$,then $Q \in S^\perp$.*
(3) *If $S_1 \subseteq S_2$, then $S_2^\perp \subseteq S_1^\perp$*
(4) $S \subseteq S^{\perp\perp}$.
(5) $S^{\perp\perp\perp} = S^\perp$
(6) *Let $\mathcal{S}$ be a collection of sets. Then, $(\bigcup \mathcal{S})^\perp = \bigcap_{S\in\mathcal{S}} S^\perp$.*
(7) *Let $\mathcal{S}$ be a collection of sets S s.t. $S = S^{\perp\perp}$, whenever $S \in \mathcal{S}$. Then, $(\bigcap \mathcal{S})^{\perp\perp} = \bigcap \mathcal{S}$.*

Proof. **(1)** Follows by Lemma D.5(1).
(2) Follows by Lemma D.5(2).
(3) Suppose $P \in S_2^\perp$.
So let $Q \in S_1$. Since $S_1 \subseteq S_2$, then $Q \in S_2$. Since $P \in S_2^\perp$, then $P \; |x| \; Q$ is SN.
Thus, $P \in S_1^\perp$.
(4) Let $P \in S$. We want $P \in S^{\perp\perp}$. Take $Q \in S^\perp$. It suffices to show that $P \; |x| \; Q$ is SN. It follows from $Q \in S^\perp$ and $P \in S$.
(5) From (2) and (3) follows $S^{\perp\perp\perp} \subseteq S^\perp$. From (3) follows $S^\perp \subseteq (S^\perp)^{\perp\perp} = S^{\perp\perp\perp}$.
(6) We prove that (i) $(\bigcup \mathcal{S})^\perp \subseteq \bigcap_{S\in\mathcal{S}} S^\perp$ and (ii) $\bigcap_{S\in\mathcal{S}} S^\perp \subseteq (\bigcup \mathcal{S})^\perp$.
(ii) follows immediately by Def. D.3.
So let us consider (i).
Let $S \in \mathcal{S}$. Applying (3) to $S \subseteq \bigcup \mathcal{S}$ yields $(\bigcup \mathcal{S})^\perp \subseteq S^\perp$.
Then, $(\bigcup \mathcal{S})^\perp \subseteq \bigcap_{S\in\mathcal{S}} S^\perp$.
(7) We have

$$
\begin{aligned}
(\textstyle\bigcap \mathcal{S})^{\perp\perp} &= (\textstyle\bigcap_{S\in\mathcal{S}} S)^{\perp\perp} \\
&= (\textstyle\bigcap_{S\in\mathcal{S}} S^{\perp\perp})^{\perp\perp} && (S = S^{\perp\perp}, \text{ whenever } S \in \mathcal{S}) \\
&= (\textstyle\bigcup_{S\in\mathcal{S}} S^{\perp})^{\perp\perp\perp} && (\text{from (6)}) \\
&= (\textstyle\bigcup_{S\in\mathcal{S}} S^{\perp})^{\perp} && (\text{from (5)}) \\
&= \textstyle\bigcap_{S\in\mathcal{S}} S^{\perp\perp} && (\text{from (6)}) \\
&= \textstyle\bigcap_{S\in\mathcal{S}} S && (S = S^{\perp\perp}, \text{ whenever } S \in \mathcal{S})
\end{aligned}
$$

D.4 Logical Predicates $[\![x : A]\!]_\sigma$

We will now introduce the logical predicates $[\![x : A]\!]_\sigma$ for strong normalisation. Since we are working with polymorphic and inductive types, the definition is parametric on a map σ from type variables to reducibility candidates. So let us define reducibility candidates first.

Definition D.4 (Reducibility Candidates $R[x : A]$). *Given a type A and a name x we define a reducibility candidate at $x : A$, denoted by $R[x : A]$ as a set of SN processes $P \vdash x : A$ which is equal to its biorthogonal, i.e. $R[x : A] = R[x : A]^{\perp\perp}$.*

We let $\mathcal{R}[- : A]$ be the set of all reducibility candidates $R[x : A]$ for some name x. Reducibility candidates are ordered by set-inclusion $\subseteq$, the least candidate being $\emptyset^{\perp\perp}$.

$$
\begin{array}{ll}
[\![x : X]\!]_\sigma & \triangleq \sigma(X)[x] \\
[\![x : \mathbf{1}]\!]_\sigma & \triangleq \{P \mid P \approx \mathsf{close}\ x \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : A \otimes B]\!]_\sigma & \triangleq \{P \mid \exists P_1, P_2.\ P \approx \mathsf{send}\ x(y.P_1); P_2 \text{ and} \\
& \qquad P_1 \in [\![y : A]\!]_\sigma \text{ and } P_2 \in [\![x : B]\!]_\sigma\}^{\perp\perp} \\
[\![x : A \oplus B]\!]_\sigma & \triangleq \{P \mid \exists Q.\ P \approx x.\mathsf{inl}; Q \text{ and } Q \in [\![x : A]\!]_\sigma \text{ or} \\
& \qquad P \approx x.\mathsf{inr}; Q \text{ and } Q \in [\![x : B]\!]_\sigma\}^{\perp\perp} \\
[\![x : !A]\!]_\sigma & \triangleq \{P \mid \exists Q.\ P \approx !x(y); Q \text{ and } Q \in [\![y : A]\!]_\sigma\}^{\perp\perp} \\
[\![x : \exists X.A]\!] & \triangleq \{P \mid \exists Q, S \in \mathcal{R}[- : B].\ P \approx \mathsf{sendty}\ x(B); Q \text{ and} \\
& \qquad Q \in [\![x : A]\!]_{\sigma[X \mapsto S]}\}^{\perp\perp} \\
[\![x : \mu X.\ A]\!]_\sigma & \triangleq (\bigcap\{S \in \mathcal{R}[- : \mu X.A] \mid \mathsf{unfold}_\mu\ x; [\![x : A]\!]_{\sigma[X \mapsto S]} \subseteq S\})^{\perp\perp} \\
[\![x : \wedge A]\!]_\sigma & \triangleq \{P \mid \exists Q.\ P \approx \mathsf{affine}\ x; Q \text{ and } Q \in [\![x : A]\!]_\sigma\}^{\perp\perp} \\
[\![x : \mathsf{S}_\bullet A]\!]_\sigma & \triangleq \{P \mid P \approx \mathsf{cell}\ x(y.[\![y : \wedge A]\!]_\sigma) \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : \mathsf{S}_\circ A]\!]_\sigma & \triangleq \{P \mid P \approx \mathsf{empty}\ x(y.[\![y : \wedge A]\!]_\sigma) \text{ and } P \text{ is SN}\}^{\perp\perp} \\
[\![x : B]\!]_\sigma & \triangleq [\![x : \overline{B}]\!]_\sigma^\perp \ \text{ (B negative type)}
\end{array}
$$

Fig. 25: Logical Predicate $[\![x : A]\!]_\sigma$.

Definition D.5 (Logical Predicate $[\![x : A]\!]_\sigma$). *By induction on the type A, we define the set $[\![x : A]\!]_\sigma$ an shown in Fig. 25. The definition is direct for the positive types A, for negative types B is simply given by orthogonality. Furthermore, we constrain the elements of $[\![x : \mathsf{U}_\bullet A]\!]_\sigma$ and $[\![x : \mathsf{U}_\circ A]\!]_\sigma$ to be $[\![y : \wedge\overline{A}]\!]$-preserving, for all y.*

For the positive types A, the predicate $[\![x : A]\!]_\sigma$ takes the biorthogonal of some base set S of processes P that offer an action, further conditions then characterise the process constituents of the actions. In the base cases $\mathsf{close}\ x$, $\mathsf{cell}\ x(y.[\![y : \wedge A]\!]_\sigma)$ and $\mathsf{empty}\ x(y.[\![y : \wedge A]\!]_\sigma)$, where the action does not have any further process constituents, we simply require the action offering process to be SN.

The presence of duality give us some succinctness in the presentation of the logical predicates, since, for the negative types A, the predicate $[\![x : A]\!]_\sigma$ is simply defined as the biorthogonal of the logical predicate for its dual $\overline{A}$ type. In fact, we can also establish this property for the positive types (Lemma D.7(4)), thereby lifting duality to the logical level using the orthogonal operation. As a pleasant consequence we conclude immediately that if $P[\![x : A]\!]_\sigma$ and $Q \in [\![x : \overline{A}]\!]_\sigma$, then the resulting cut composition $P\ |x|\ Q$ is SN.

By exploiting the properties satisfied by the orthogonal (Lemma D.6) we obtain a strategy to establish the membership $P \in [\![x : A]\!]_\sigma$. For the positive types we have $[\![x : A]\!]_\sigma = S^{\perp\perp}$, for some set S. Since $S \subseteq S^{\perp\perp}$ (Lemma D.6(4)), we can conclude that $P \in [\![x : A]\!]_\sigma$, provided we prove $P \in S$. On the other hand, for the negative types we have $[\![x : A]\!]_\sigma = S^{\perp\perp\perp}$. But since $S^{\perp\perp\perp} = S^\perp$ (Lemma D.6(5)), it is equivalent to prove that for all $Q \in S$, $P\ |x|\ Q$ is SN. These strategies will be applied throughout the proof of the Fundamental Lemma D.11.

In all cases, with some exceptions, when defining $[\![x : A]\!]_\sigma$ we simply propagate map σ without modifications. The exceptions are the defining clauses corresponding to the existential $\exists X.A$ and the inductive types $\mu X.\ A$, in which we extend the map σ with an assignment for the type variable X. Furthermore, the definition of the predicate for a type variable $[\![x : X]\!]_\sigma$ picks the corresponding reducibility candidate $\sigma(X) = R[y : B]$, instantiated at name x: $\{x/y\}R[y : B]$.

The definition of $[\![x : \mu X.\ A]\!]_\sigma$ relies on the construction $\mathsf{unfold}_\mu\ x; S$, that for any set S, is defined according to

$$\mathsf{unfold}_\mu\ x; S \triangleq \{P \mid \exists Q.\ P \approx \mathsf{unfold}_\mu\ x; Q \text{ and } \ Q \in S\}$$

Similarly, given a set S, we define $\mathsf{unfold}_\nu\ x; A$ by

$$\mathsf{unfold}_\nu\ x; S \triangleq \{P \mid \exists Q.\ P \approx \mathsf{unfold}_\nu\ x; Q \text{ and } \ Q \in S\}$$

The following lemma states some basic properties about the logical predicates.

Lemma D.7 (Logical Predicates: Basic Properties). *The following properties hold*

(1) *If* $P \in [\![x : A]\!]_\sigma$, *then* $\{y/x\}P \in [\![y : A]\!]_\sigma$.
(2) *If* $P \in [\![x : A]\!]_\sigma$ *and* $P \approx Q$, *then* $Q \in [\![x : A]\!]_\sigma$.
(3) *If* $P \in [\![x : A]\!]_\sigma$ *and* $P \to Q$, *then* $Q \in [\![x : A]\!]_\sigma$.
(4) $[\![x : \overline{A}]\!]_\sigma \ = [\![x : A]\!]_\sigma^\perp$.
(5) $[\![x : \{B/X\}A]\!]_\sigma = [\![x : A]\!]_{\sigma[X \mapsto [\![x:B]\!]_\sigma]}$.
(6) $[\![x : A]\!]_{\sigma[X \mapsto S^\perp]} = [\![x : \{\overline{X}/X\}A]\!]_{\sigma[X \mapsto S]}$.

Proof. Property (1) is trivial. Properties (2) and (3) follows by Lemma D.6(1) and Lemma D.6(2), respectively. Property (4) follows directly by Def. D.5 for half of the types. The remaining half follows by Lemma D.6(5). Properties (5) and (6) are straightforward by induction on A.

The logical predicates are preserved by name substitution, the congruence relation $\approx$ and the reduction relation $\to$ (Lemma D.7(1)-(3)). Property Lemma D.7(4) relates the logical predicates of duality related types, using the orthogonal. Lemma D.7(5)-(6) relate type variable substitution with the parametric map σ.

We use the interference-sensitive reference cells (Def. D.1) to define the logical predicates $[\![x : \mathsf{S}_\bullet A]\!]_\sigma$ and $[\![c : \mathsf{S}_\circ A]\!]_\sigma$, for the state full and the state empty modalities, respectively. This allows us to internalise state interference in the definition of the logical predicate itself and, as consequence, we can reason modularly about state sharing as witnessed by the following lemma

Lemma D.8. *The following properties hold*

(1) *If* $P_1 \in [\![c : \mathsf{U}_\bullet A]\!]_\sigma$ *and* $P_2 \in [\![c : \mathsf{U}_\bullet A]\!]_\sigma$, *then* $\mathsf{share}\ c\ \{P_1 \mid\mid P_2\} \ \in [\![c : \mathsf{U}_\bullet A]\!]_\sigma$.

(2) *If* $P_1 \in [\![c : \mathsf{U}_\circ A]\!]_\sigma$ *and* $P_2 \in [\![c : \mathsf{U}_\circ A]\!]_\sigma$*, then* $\mathsf{share}\ c\ \{P_1 \mid\mid P_2\} \in [\![c : \mathsf{U}_\circ A]\!]_\sigma$*.*

Proof. **(1)** By Def. D.3 and Lemma D.6(5) we have $[\![c : \mathsf{U}_\bullet A]\!] = S^\perp$, where

$$S = \{Q \mid Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\}.$$

Let $Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma$.
We need to prove that $Q\ |c|\ \mathsf{share}\ c\ \{P_1 \mid\mid P_2\}$ is SN.
By Lemma D.4(1) we conclude that $Q\ |c|\ \mathsf{share}\ c\ \{P_1 \mid\mid P_2\}$ is simulated by

$$(Q\ |c|\ P_1) \mid\mid (Q\ |c|\ P_2)$$

By hypothesis, $P_1 \in [\![c : \mathsf{U}_\bullet A]\!]_\sigma$, hence $Q\ |c|\ P_1$ is SN.
By hypothesis, $P_2 \in [\![c : \mathsf{U}_\bullet A]\!]_\sigma$, hence $Q\ |c|\ P_2$ is SN.
Then, $(Q\ |c|\ P_1) \mid\mid (Q\ |c|\ P_2)$ is SN (Lemma D.5(4)).
Therefore, $Q\ |c|\ \mathsf{share}\ c\ \{P_1 \mid\mid P_2\}$ is SN (Lemma D.5(5)).
By hypothesis, for any y, both P_1 and P_2 are $[\![y : \wedge\overline{A}]\!]$-preserving on c. Applying Lemma D.3, we conclude that $\mathsf{share}\ c\ \{P_1 \mid\mid P_2\}$ is also $[\![y : \wedge\overline{A}]\!]$-preserving on c.

(2) Similarly to (1), by applying the simulation Lemma D.4(2).

We will now state some properties concerning the logical predicate for inductive types. But first, let us introduce the following definition.

Definition D.6 **($\phi_A(S)$).** *Suppose that X occurs positively on A. Define*

$$\phi_A(S) \triangleq \mathsf{unfold}_\mu\ x; [\![x : A]\!]_{\sigma[X \mapsto S]}$$

$[\![x : \mu X.\ A]\!]_\sigma$ is defined as the biorthogonal of the intersection of all ϕ_A-closed sets S, i.e. sets S s.t. $\phi_A(S) \subseteq S$. Since ϕ_A is monotonic (Lemma D.9(1)), Knaster-Tarski theorem implies that $[\![x : \mu X.\ A]\!]_\sigma$ is the least fixed point of ϕ_A (Lemma D.9(2)). Dually, we can obtain a greatest fixed point characterisation for $[\![x : \nu X.\ A]\!]_\sigma$ (Lemma D.9(3)). Applying Kleene's fixed point theorem we explicitly construct the fixed point of ϕ_A (Lemma D.9(4)).

Lemma D.9. *The following properties hold*

(1) *The map ϕ_A is monotonic, i.e. $\phi_A(S_1) \subseteq \phi_A(S_2)$, whenever $S_1 \subseteq S_2$.*
(2) $[\![x : \mu X.\ A]\!]_\sigma$ *is the least fixed point of ϕ_A.*
(3) *Let $\psi_A(S) \triangleq \phi_{\{\overline{X}/X\}\overline{A}}(S^\perp)^\perp$. Then, $[\![x : \nu X.\ A]\!]_\sigma$ is the greatest fixed point of ψ_A.*
(4) $[\![x : \mu X.\ A]\!]_\sigma = \bigcup_{n \in \mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$.
(5) $\mathsf{unfold}_\nu\ x; [\![x : \{\nu X.\ A/X\}A]\!]_\sigma \subseteq [\![x : \nu X.\ A]\!]_\sigma$.

Proof. **(1)** We prove hypothesis (H1) if $S_1 \subseteq S_2$, then $[\![x : A]\!]_{\sigma[X \mapsto S_1]} \subseteq [\![x : A]\!]_{\sigma[X \mapsto S_2]}$, which implies (1).
The proof of (H1) is by induction on A, we handle some representative cases.

Case: $A = Y$.
There are two cases to consider, depending on whether (i) $Y \neq X$ or (ii) $Y = X$.
If (i), then $[\![x : Y]\!]_{\sigma[X \mapsto S_1]} = \sigma(Y) = [\![x : Y]\!]_{\sigma[X \mapsto S_2]}$.
If (ii), then $[\![x : X]\!]_{\sigma[X \mapsto S_1]} = S_1 \subseteq S_2 = [\![x : X]\!]_{\sigma[X \mapsto S_2]}$.
In either case (i)-(ii), $[\![x : Y]\!]_{\sigma[X \mapsto S_1]} \subseteq [\![x : Y]\!]_{\sigma[X \mapsto S_2]}$.

Case: $A = \mathbf{1}$.
We have $[\![x : \mathbf{1}]\!]_{\sigma[X \mapsto S_1]} = [\![x : \mathbf{1}]\!]_{\sigma[X \mapsto S_2]}$.

Case: $A = A_1 \otimes A_2$.
By Def. D.5,

$$[\![x : A_1 \otimes A_2]\!]_{\sigma[X \mapsto S]} = f(S)^{\perp\perp}$$

where

$$\begin{aligned} f(S) \triangleq \{P \mid \exists P_1, P_2.\ & P \approx \mathsf{send}\ x(y.P_1); P_2 \\ & \text{and } P_1 \in [\![y : A_1]\!]_{\sigma[X \mapsto S]} \text{ and } P_2 \in [\![x : A_2]\!]_{\sigma[X \mapsto S]}\} \end{aligned}$$

Suppose that $S_1 \subseteq S_2$. I.h. applied to A_1 and A_2 yields $f(S_1) \subseteq f(S_2)$.
Lemma D.6(3) applied twice to $f(S_1) \subseteq f(S_2)$ yields

$$[\![x : A_1 \otimes A_2]\!]_{\sigma[X \mapsto S_1]} = f(S_1)^{\perp\perp} \subseteq f(S_2)^{\perp\perp} = [\![x : A_1 \otimes A_2]\!]_{\sigma[X \mapsto S_2]}$$

Case: $A = \mu Y.\ B$.
By Def. D.5

$$[\![x : \mu Y.\ B]\!]_{\sigma[X \mapsto S]} = (\bigcap f(S))^{\perp\perp}$$

where

$$f(S) \triangleq \{T \in \mathcal{R}[- : \mu Y.B] \mid \mathsf{unfold}_\mu\ x; [\![x : B]\!]_{\sigma[X \mapsto S, Y \mapsto T]} \subseteq T\}$$

Suppose $S_1 \subseteq S_2$. Let $T \in f(S_2)$. Then, $\mathsf{unfold}_\mu\ x; [\![x : B]\!]_{\sigma[X \mapsto S_2, Y \mapsto T]} \subseteq T$.
I.h. applied to B yields $\mathsf{unfold}_\mu\ x; [\![x : B]\!]_{\sigma[X \mapsto S_1, Y \mapsto T]} \subseteq \mathsf{unfold}_\mu\ x; [\![x : B]\!]_{\sigma[X \mapsto S_2, Y \mapsto T]}$.
By transitivity of $\subseteq$, $\mathsf{unfold}_\mu\ x; [\![x : B]\!]_{\sigma[X \mapsto S_1, Y \mapsto T]} \subseteq T$.
Hence, $T \in f(S_1)$.
This establishes $f(S_2) \subseteq f(S_1)$.
Then, $\bigcap f(S_1) \subseteq \bigcap f(S_2)$.
Lemma D.6(3) applied twice to $\bigcap f(S_1) \subseteq \bigcap f(S_2)$ yields

$$[\![x : \mu Y.\ B]\!]_{\sigma[X \mapsto S_1]} = (\bigcap f(S_1))^{\perp\perp} \subseteq (\bigcap f(S_2))^{\perp\perp} = [\![x : \mu Y.\ B]\!]_{\sigma[X \mapsto S_2]}$$

Case: $A = \mathsf{S}_\bullet B$.
By Def. D.5

$$[\![x : \mathsf{S}_\bullet B]\!]_{\sigma[X \mapsto S]} = f(S)^{\perp\perp}$$

where

$$f(S) \triangleq \{P \mid P \approx \mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S]})\}$$

Suppose $S_1 \subseteq S_2$. We prove that $f(S_2)^\perp \subseteq f(S_1)^\perp$.
Let $Q \in f(S_2)^\perp$. In order to show that $Q \in f(S_1)^\perp$ we must show that $P \mid x \mid Q$ is SN, when $P \in f(S_1)$.
We prove by induction on $N(P)+N(Q)$ that all the reductions $P \mid x \mid Q \to R$ are SN.
We handle only the interesting reduction, which corresponds to a cell-take interaction on session x. Then

$$\begin{aligned} P \mid x \mid Q &\approx \mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_1]}) \mid x \mid \mathsf{take}\ x(y); Q' \\ &\to \mathsf{empty}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_1]}) \mid x \mid (P' \mid y \mid Q') = R \end{aligned}$$

where $P \approx \mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_1]})$, $Q \approx \mathsf{take}\ x(y); Q'$ and P' is some element in $[\![y : \wedge A]\!]_{\sigma[X \mapsto S_1]}$. By hypothesis, $Q \in f(S_2)^\perp$, hence

$$\mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_2]}) \mid x \mid \mathsf{take}\ x(y); Q'$$

is SN.
Then, all the reductions of $\mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_2]}) \mid x \mid \mathsf{take}\ x(y); Q'$ are SN, in particular the following reduction can be obtained, since $P' \in S_1 \subseteq S_2$:

$$\begin{aligned} &\mathsf{cell}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_2]}) \mid x \mid \mathsf{take}\ x(y); Q' \\ &\to \mathsf{empty}\ x(y.[\![y : \wedge A]\!]_{\sigma[X \mapsto S_2]}) \mid x \mid (P' \mid y \mid Q') \end{aligned}$$

(2) By Def. D.5

$$[\![x : \mu X.\ A]\!]_\sigma = \left(\bigcap \{S \in \mathcal{R}[- : \mu X.A] \mid \phi_A(S) \subseteq S\}\right)^{\perp\perp}$$

Since a reducibility candidate is equal to its biorthogonal (Def. D.4), we can write $[\![x : \mu X.\ A]\!]_\sigma$ in the alternative form (Lemma D.6(7))

$$[\![x : \mu X.\ A]\!]_\sigma = \bigcap \{S \in \mathcal{R}[- : \mu X.A] \mid \phi_A(S) \subseteq S\}$$

i.e. $[\![x : \mu X.\ A]\!]_\sigma$ is the intersection of all ϕ_A-closed sets in $\mathcal{R}[- : \mu X.A]$.
We now prove the following propositions

(i) $[\![x : \mu X.\ A]\!]_\sigma$ is ϕ_A-closed, i.e. $\phi_A([\![x : \mu X.\ A]\!]_\sigma) \subseteq [\![x : \mu X.\ A]\!]_\sigma$.
Let $S \in \mathcal{R}[- : \mu X.A]$ be a ϕ_A-closed set.
By definition, we have (a) $\phi_A(S) \subseteq S$ and (b) $[\![x : \mu X.\ A]\!]_\sigma \subseteq S$.
Monotonicity of ϕ_A (1) applied to (b) yields $\phi_A([\![x : \mu X.\ A]\!]_\sigma \subseteq \phi_A(S)$.
Hence, transitivity and (a) implies $\phi_A([\![x : \mu X.\ A]\!]_\sigma) \subseteq S$.
Since $[\![x : \mu X.\ A]\!]_\sigma$ is the intersection of all ϕ_A-closed sets in $\mathcal{R}[- : \mu X.A]$, then $\phi_A([\![x : \mu X.\ A]\!]_\sigma) \subseteq [\![x : \mu X.\ A]\!]_\sigma$.

(ii) $[\![x : \mu X.\ A]\!]_\sigma \subseteq \phi_A([\![x : \mu X.\ A]\!]_\sigma)$.
Monotonicity of ϕ_A (1) applied to (i) yields $\phi_A(\phi_A([\![x : \mu X.\ A]\!]_\sigma) \subseteq \phi_A([\![x : \mu X.\ A]\!]_\sigma)$, i.e. $\phi_A([\![x : \mu X.\ A]\!]_\sigma)$ is ϕ_A-closed.
Since $[\![x : \mu X.\ A]\!]_\sigma$ is the intersection of all ϕ_A-closed sets in $\mathcal{R}[- : \mu X.A]$, then $[\![x : \mu X.\ A]\!]_\sigma \subseteq \phi_A([\![x : \mu X.\ A]\!]_\sigma)$.

Propositions (i) and (ii) imply that $[\![x : \mu X.\ A]\!]_\sigma$ is a fixed point of ϕ_A.
Let $S \in \mathcal{R}[- : \mu X.A]$ be any fixed point of ϕ_A. Then, in particular, S is ϕ_A-closed, hence $[\![x : \mu X.\ A]\!]_\sigma \subseteq \phi_A$.
Therefore, $[\![x : \mu X.\ A]\!]_\sigma$ is the least fixed point of ϕ_A.

(3) We need to prove the following propositions

(i) $[\![x : \nu X.\ XA]\!]_\sigma$ is a fixed point of ψ_A.
By (b), $[\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma$ is a fixed point of $\phi_{\{\overline{X}/X\}\overline{A}}$

$$\phi_{\{\overline{X}/X\}\overline{A}}([\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma) = [\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma$$

hence, applying the orthogonal to both sides of the equation yields

$$\phi_{\{\overline{X}/X\}\overline{A}}([\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma)^\perp = [\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma^\perp$$

Since $[\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma^\perp = [\![x : \nu X.\ XA]\!]_\sigma$ (Lemma D.7(4)) we can rewrite the equation in the equivalent form

$$\phi_{\{\overline{X}/X\}\overline{A}}([\![x : \nu X.\ XA]\!]_\sigma^\perp)^\perp = [\![x : \nu X.\ XA]\!]_\sigma$$

Then, $[\![x : \nu X.\ XA]\!]_\sigma$ is a fixed point of ψ_A.

(ii) If S is a fixed point of ψ_A, then $S \subseteq [\![x : \nu X.\ XA]\!]_\sigma$.
Suppose that S is a fixed point of ψ_A, i.e.

$$\psi_A(S) = \phi_{\{\overline{X}/X\}\overline{A}}(S^\perp)^\perp = S$$

Applying the orthogonal to both sides of the equation yields

$$\phi_{\{\overline{X}/X\}\overline{A}}(S^\perp)^{\perp\perp} = S^\perp$$

Since $\phi_{\{\overline{X}/X\}\overline{A}}(S^\perp) \subseteq \phi_{\{\overline{X}/X\}\overline{A}}(S^\perp)^{\perp\perp}$ (Lemma D.6(4)), then

$$\phi_{\{\overline{X}/X\}\overline{A}}(S^\perp) \subseteq S^\perp$$

i.e. $S^\perp$ is a $\phi_{\{\overline{X}/X\}\overline{A}}$-closed set.
Then, by Def. D.5

$$[\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma \subseteq S^\perp$$

Applying the orthogonal to the inequation (Lemma D.6(3)) yields

$$S^{\perp\perp} \subseteq [\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma^\perp$$

Since $S \subseteq S^{\perp\perp}$ (Lemma D.6(2)), we obtain

$$S \subseteq [\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma^\perp$$

Finally, since $[\![x : \mu X.\ \{\overline{X}/X\}\overline{A}]\!]_\sigma^\perp = [\![x : \nu X.\ A]\!]_\sigma$ (Lemma D.7(4)) we have

$$S \subseteq [\![x : \nu X.\ A]\!]_\sigma$$

(4) We prove that $\bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$ is the least fixed point of ϕ_A.
By (b) it follows that $[\![x : \mu X.\ A]\!]_\sigma = \bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$.
We need to prove the following propositions

(i) $\bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$ is a fixed point of ϕ_A.
We have

$$\phi_A(\bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})) = \bigcup_{n>0} \phi_A^n(\emptyset^{\perp\perp})$$

Since $\phi_A^0(\emptyset^{\perp\perp}) = \emptyset^{\perp\perp}$ is the least reducibility candidate, we have $\phi_A^0(\emptyset^{\perp\perp}) \subseteq \phi_A^n(\emptyset^{\perp\perp})$, for any $n > 0$.
Then

$$\bigcup_{n>0} \phi_A^n(\emptyset^{\perp\perp}) = \bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$$

Therefore

$$\phi_A(\bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})) = \bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp})$$

(ii) If S is fixed point of ϕ_A, then $\bigcup_{n\in\mathbb{N}} \phi_A^n(\emptyset^{\perp\perp}) \subseteq S$.
We that $\phi_A^n(\emptyset^{\perp\perp}) \subseteq S$, for all $n \in \mathbb{N}$. The proof is by induction on n.

Case: $n = 0$.
Since $\phi_A^0(\emptyset^{\perp\perp}) = \emptyset^{\perp\perp}$ is the least reducibility candidate, $\phi_A^0(\emptyset^{\perp\perp}) \subseteq S$.

Case: $n = m + 1$.
By i.h. we have

$$\phi_A^m(\emptyset^{\perp\perp}) \subseteq S$$

Monotonicity of ϕ_A (1) implies

$$\phi_A^{m+1}(\emptyset^{\perp\perp})) \subseteq \phi(S)$$

Since $\phi(S) = S$, then

$$\phi_A^{m+1}(\emptyset^{\perp\perp})) \subseteq S$$

(5) Let $P \approx \mathsf{unfold}_\nu\ x; P'$, where $P' \in [\![x : \{\nu X.\ A/X\}A]\!]_\sigma$.
Let $B \triangleq \{\overline{X}/X\}\overline{A}$, hence $\overline{\nu X.\ A} = \mu X.\ B$.
We prove that $P\ |x|\ Q$ is SN, for all $Q \in [\![x : \mu X.\ B]\!]_\sigma$, by analysing all the possible reductions of $P\ |x|\ Q$ and concluding that all of them are SN.
The critical reduction is the unfold-unfold interaction on session x, in which case $Q \approx \mathsf{unfold}_\nu\ x; Q'$, and $P\ |x|\ Q \to P'\ |x|\ Q'$.
By (2) we conclude that $Q' \in [\![x : \{\mu X.\ B/X\}B]\!]_\sigma$.
Since $\overline{\{\nu X.\ A/X\}A} = \{\mu X.\ B/X\}B$, we conclude that $P'\ |x|\ Q'$ is SN.

D.5 Extended Logical Predicate and Fundamental Lemma

The logical predicates $[\![x : A]\!]_\sigma$ introduced previously apply to processes that have a single free name x. We will now extend the definition to typed processes $P \vdash_\eta \Delta; \Gamma$ with an arbitrary set of free names. The idea is to compose

P with candidates from the basic logical predicates and require the composition to be strongly normalising. We then conclude with the statement and proof of the Fundamental Lemma D.11, from which strong normalisation for $\to$ follows (Theorem 3.3). Let us start with the following definition.

Definition D.7 (Logical Contexts). *The set* $[\![\Delta]\!]_\sigma$ *of linear logical contexts at* Δ *is inductively defined by*

$$[\![\emptyset]\!]_\sigma \triangleq \{-\} \quad [\![\Delta, x : A]\!]_\sigma \triangleq \{P \mid x : \overline{A} \mid \mathcal{C} \mid P \in [\![x : \overline{A}]\!]_\sigma \text{ and } \mathcal{C} \in [\![\Delta]\!]_\sigma\}$$

Similarly, we define the set $[\![\Gamma]\!]^!_\sigma$ *of unrestricted logical contexts at* Γ *inductively by*

$$[\![\emptyset]\!]^!_\sigma \triangleq \{-\} \quad [\![\Gamma, y : A]\!]^!_\sigma \triangleq \{y.P \mid !x : \overline{A} \mid \mathcal{C} \mid P \in [\![y : \overline{A}]\!]_\sigma \text{ and } \mathcal{C} \in [\![\Gamma]\!]^!_\sigma\}$$

We extend N from processes to contexts $\mathcal{C} \in [\![\Delta]\!]_\sigma$ by $N(-) = 0$ and $N(P \mid x \mid \mathcal{C}') = N(P) + N(\mathcal{C}')$. Now, we will extend the logical predicate to general typed processes $P \in [\![\vdash_\eta \Delta; \Gamma]\!]$ by composing it along Δ and Γ with processes from the basic logical predicates (Def. D.5) and by replacing the free process variables by elements of the appropriate reducibility candidate, according to the following definition.

Definition D.8 (Extended Logical Predicate $[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$). *We define* $\mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$ *inductively on* η *as the set of processes* $P \vdash_\eta \Delta; \Gamma$ *s.t.*

$$\begin{array}{ll} P \in \mathcal{L}[\![\vdash_\emptyset \Delta; \Gamma]\!]_\sigma & \textit{iff } \forall \mathcal{C} \in [\![\Delta]\!]_\sigma \ \forall \mathcal{D} \in [\![\Gamma]\!]^!_\sigma.\ \mathcal{C} \circ \mathcal{D}[P] \textit{ is SN.} \\ P \in \mathcal{L}[\![\vdash_{\eta, X(x,\vec{y}) \mapsto \Delta', x:Y; \Gamma} \Delta; \Gamma]\!]_\sigma & \textit{iff } \forall Q \in \mathcal{L}[\![\vdash_\emptyset \Delta', x : Y; \Gamma]\!].\ \{Q/X\}P \in \mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma. \end{array}$$

The base case $\mathcal{L}[\![\vdash_\emptyset \emptyset; \emptyset]\!]_\sigma$ corresponds to the set of closed typed SN processes. Given a map

$$\eta = X_1(\vec{x_1}) \mapsto \Delta_1; \Gamma_1, \ldots, X_n(\vec{x_n}) \mapsto \Delta_n; \Gamma_n$$

we define $[\![\eta]\!]_\sigma$ as the set of all substitution maps η' s.t.

$$\eta' = X_1(\vec{x_1}) \mapsto Q_1, \ldots, X_n(\vec{x_n}) \mapsto Q_n$$

where $Q_1 \in \mathcal{L}[\![\vdash_\emptyset \Delta_1; \Gamma_1]\!]_\sigma, \ldots, Q_n \in \mathcal{L}[\![\vdash_\emptyset \Delta_n; \Gamma_n]\!]_\sigma$.

Then, Def. D.8 is equivalent to the following

$$P \in \mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma \text{ iff } \forall \eta' \in [\![\eta]\!]_\sigma \ \forall \mathcal{C} \in [\![\Delta]\!]_\sigma \ \forall \mathcal{D} \in [\![\Gamma]\!]^!_\sigma.\ \mathcal{C} \circ \mathcal{D}[\eta'(P)] \text{ is SN.}$$

where we denote by $\eta'(P)$ the process obtained by substituting the variables in P by processes according to η'.

The following property establishes an equivalence between the extended logical predicate and the basic logical predicates of Def. D.5. In one direction it establishes that if $P \in \mathcal{L}[\![\vdash_\emptyset \Delta, x : A; \Gamma]\!]_\sigma$, then we can cut the process along Δ and Γ and prove that the resulting cut composition is an element of $[\![x : A]\!]_\sigma$.

Lemma D.10. *The following two propositions*

(1) $P \in \mathcal{L}[\![\vdash_\emptyset \Delta, x : A; \Gamma]\!]_\sigma$.
(2) *For all* $\mathcal{C} \in [\![\Delta]\!]_\sigma$ *and* $\mathcal{D} \in [\![\Gamma]\!]^!{}_\sigma$, $\mathcal{C} \circ \mathcal{D}[P] \in [\![x : A]\!]_\sigma$.

are equivalent.

Proof. By Lemma D.7(4).

Lemma D.10 gives us a degree of freedom in the sense that we can choose an arbitrary typed channel $x : A$ from a nonempty linear typing context Δ of a typed process $P \vdash_\emptyset \Delta; \Gamma$ and cut the remaining linear context. We conclude this section with the proof of the Fundamental Lemma D.11, from which strong normalisation (Theorem 3.3) follows immediately.

Lemma D.11 (Fundamental Lemma). *If* $P \vdash_\eta \Delta; \Gamma$, *then* $P \in \mathcal{L}[\![\vdash_\eta \Delta; \Gamma]\!]_\sigma$ *for all* σ.

Proof. By induction on the structure of a typing derivation for $P \vdash_\eta \Delta; \Gamma$. Cases [Tcut], [Tfwd], [Tcut!] follow immediately because $[\![x : A]\!] = [\![x : \overline{A}]\!]^\perp$. Case [T0] follows because 0 is SN and case [Tmix] follows because $P \mid\mid Q$ is SN whenever P and Q are SN. For the positive types A, the logical predicate $[\![x : A]\!]_\sigma$ is defined as the biorthogonal of some set S, hence for the typing rules that introduce a positive type A the strategy is to show that the introduced action P lies in $S \subseteq S^{\perp\perp}$. For the negative types $\overline{A}$: $[\![x : \overline{A}]\!]_\sigma = S^{\perp\perp\perp} = S^\perp$, hence the strategy for the typing rules that that introduce an action Q that types with a negative type $x : \overline{A}$ is to show that $P \mid x : A \mid Q$ is SN, for all $P \in S$. Particularly, for rule [Tcorec], where $A = \mu X.\ B$, we proceed by induction on the depth n of unfolding, since $S \bigcup_{n \in \mathbb{N}} \phi_B^n(\emptyset^{\perp\perp})$. Cases [Tcell] and [Tempty] follow by applying the simulations Lemma D.2(1)-(2). Cases [Tsh], [TshL], [TshR] follows after applying the *decomposition of the share as a mix* as given by Lemma D.4(1)-(2). We illustrate the proof with some cases. In the cases in which the recursive map η that annotates the typing judgments $P \vdash_\eta \Delta; \Gamma$ plays no role and is essentially propagated from the conclusion to the premises of the typing rule we omit it, working as if the process P did not have any free recursion variable X. Similarly for the map σ which annotates the logical predicates $[\![x : A]\!]_\sigma$.

Case: [T0]:

$$\frac{}{0 \vdash \cdot; \Gamma}$$

Let $\mathcal{C}_! \in [\![\Gamma]\!]^!$.
Then, $\mathcal{C}_![0]$ is SN.

Case [Tmix]:

$$\frac{P_1 \vdash \Delta_1; \Gamma \quad P_2 \vdash \Delta_2; \Gamma}{P_1 \mid\mid P_2 \vdash \Delta_1, \Delta_2; \Gamma}$$

Let $\mathcal{C} \in [\![\Delta_1, \Delta_2]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$.
We have

$$\mathcal{C} \circ \mathcal{D}[P_1 \mid\mid P_2] \approx (\mathcal{C}_1 \circ \mathcal{D}[P_1]) \mid\mid (\mathcal{C}_2 \circ \mathcal{D}[P_2])$$

where $\mathcal{C}_1 \in [\![\Delta_1]\!]$ and $\mathcal{C}_2 \in [\![\Delta_2]\!]$.
I.h. applied to $P_1 \vdash \Delta_1; \Gamma$ yields $\mathcal{C}_1 \circ \mathcal{D}[P_1]$ is SN.
I.h. applied to $P_2 \vdash \Delta_2; \Gamma$ yields $\mathcal{C}_2 \circ \mathcal{D}[P_2]$ is SN.
By applying Lemma D.5(4) we conclude that $(\mathcal{C}_1 \circ \mathcal{D}[P_1]) \mid\mid (\mathcal{C}_2 \circ \mathcal{D}[P_2])$ is SN.
Hence, $\mathcal{C} \circ \mathcal{D}[P_1 \mid\mid P_2]$ is SN.

Case [Tfwd]:

$$\frac{}{\mathsf{fwd}\ x\ y \vdash x : A, y : \overline{A}; \Gamma}$$

Let $\mathcal{C} \in [\![x : A, y : \overline{A}]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$.
We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{fwd}\ x\ y] \approx \mathcal{D}[P\ |x|\ (Q\ |y|\ \mathsf{fwd}\ x\ y)]$$

where $P \in [\![x : \overline{A}]\!]$ and $Q \in [\![y : A]\!]$.
We prove that (H) $\mathcal{D}[P\ |x|\ (Q\ |y|\ \mathsf{fwd}\ x\ y)]$ is SN, by induction on $N(P) + N(Q)$. Suppose that $\mathcal{D}[P\ |x|\ (Q\ |y|\ \mathsf{fwd}\ x\ y)] \to R$. There are two cases to consider:

Case: (i) R is obtained by an internal reduction of either P or Q.

Case: (ii) R is obtained by an interaction with with the forwarder $\mathsf{fwd}\ x\ y$ on either session x or y.

Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Suppose w.l.o.g. that R is obtained by an interaction with the forwarder $\mathsf{fwd}\ x\ y$ on session y. Then $R \approx \mathcal{D}[P\ |x|\ \{x/y\}Q]$.
By Lemma D.7(1), $\{x/y\}Q \in [\![x : A]\!]$.
By Lemma D.7(4), $P\ |x|\ \{x/y\}Q$ is SN.
By Lemma D.5(3), $P\ |x|\ (Q\ |y|\ \mathsf{fwd}\ x\ y)$ is SN.
Hence, $\mathcal{C} \circ \mathcal{D}[\mathsf{fwd}\ x\ y]$ is SN.

Case [Tcut]:

$$\frac{P_1 \vdash \Delta_1, x : \overline{A}; \Gamma \quad P_2 \vdash \Delta_2, x : A; \Gamma}{P_1\ |x|\ P_2\ \vdash \Delta_1, \Delta_2; \Gamma}$$

Let $\mathcal{C}_1 \in [\![\Delta_1]\!]$, $\mathcal{C}_2 \in [\![\Delta_2]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$.
We have

$$\mathcal{C}_1 \circ \mathcal{C}_2 \circ \mathcal{D}[P_1\ |x|\ P_2] \approx (\mathcal{C}_1 \circ \mathcal{D}[P_1])\ |x|\ (\mathcal{C}_2 \circ \mathcal{D}[P_2])$$

I.h. and Lemma D.10 applied to $P_1 \vdash \Delta_1, x : \overline{A}; \Gamma$ yields $\mathcal{C}_1 \circ \mathcal{D}[P_1] \in [\![x : \overline{A}]\!]$.
I.h. and Lemma D.10 applied to $P_2 \vdash \Delta_2, x : A; \Gamma$ yields $\mathcal{C}_2 \circ \mathcal{D}[P_2] \in [\![x : A]\!]$.
By applying Lemma D.7(4) we conclude that $(\mathcal{C}_1 \circ \mathcal{D}[P_1])\ |x|\ (\mathcal{C}_2 \circ \mathcal{D}[P_2])$ is SN.
Hence, $\mathcal{C} \circ \mathcal{D}[P_1\ |x|\ P_2]$ is SN.

Case [Tcut!]:

$$\frac{P_1 \vdash y : \overline{A}; \Gamma \quad P_2 \vdash \Delta; \Gamma, x : A}{y.P_1 \;|!x|\; P_2 \vdash \Delta; \Gamma}$$

Let $\mathcal{C} \in [\![\Delta]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$.
We have

$$\mathcal{C} \circ \mathcal{D}[y.P_1 \;|!x|\; P_2] \approx \mathcal{C} \circ (y.\mathcal{D}[P_1] \;|!x|\; \mathcal{D})[P_2]$$

I.h. and Lemma D.10 applied to $P_1 \vdash y : \overline{A}; \Gamma$ yields $\mathcal{D}[P_1] \in [\![y : \overline{A}]\!]$. By def. D.7, $y.\mathcal{D}[P_1] \;|!x|\; \mathcal{D} \in [\![\Gamma, x : A]\!]^!$. I.h. applied to $P_2 \vdash \Delta; \Gamma, x : A$ yields $\mathcal{C} \circ (y.\mathcal{D}[P_1] \;|!x|\; \mathcal{D})[P_2]$ is SN. Hence, $\mathcal{C} \circ \mathcal{C}_![y.P_1 \;|!x|\; P_2]$ is SN.

Case [Tvar]:

$$\frac{\eta = \eta', X(x, \vec{y}) \;\mapsto \Delta, x : Y; \Gamma}{X(z, \vec{w}) \vdash_\eta \{\vec{w}/\vec{y}\}(\Delta, z : Y; \Gamma)}$$

Let $\rho \in [\![\eta]\!]_\sigma$. Then, $\rho = \rho', X(x, \vec{y}) \mapsto Q$ where $Q \in \mathcal{L}[\![\vdash_\emptyset \Delta, x : Y; \Gamma]\!]_\sigma$ and $\rho' \in [\![\eta']\!]_\sigma$.
We have

$$\rho(X(z, \vec{w})) = \{z/x\}\{\vec{w}/\vec{y}\}Q$$

Since $Q \in \mathcal{L}[\![\vdash_\emptyset \Delta, x : Y; \Gamma]\!]_\sigma$, then $\{z/x\}\{\vec{w}/\vec{y}\}Q \in \mathcal{L}[\![\vdash_\emptyset \{\vec{w}/\vec{y}\}(\Delta, z : Y; \Gamma)]\!]$. Hence, $X(z, \vec{w}) \in \mathcal{L}[\![\vdash_\eta \{\vec{w}/\vec{y}\}(\Delta, z : Y; \Gamma)]\!]$.

Case [Tsh]:

$$\frac{P_1 \vdash_\eta \Delta_1, c : \mathsf{U}_\bullet A; \Gamma \quad P_2 \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P \;||\; Q\} \;\vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\bullet A; \Gamma}$$

Let $\mathcal{C}_1 \in [\![\Delta_1]\!]$, $\mathcal{C}_2 \in [\![\Delta_2]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$.
We have

$$\begin{aligned}&\mathcal{C}_1 \circ \mathcal{C}_2 \circ \mathcal{D}[\mathsf{share}\ c\ \{P_1 \;||\; P_2\}]\\ &\approx \mathsf{share}\ c\ \{\mathcal{C}_1 \circ \mathcal{D}[P_1] \;||\; \mathcal{C}_2 \circ \mathcal{D}[P_2]\}\end{aligned}$$

I.h. and Lemma D.10 applied to $P_1 \vdash_\eta \Delta_1, c : \mathsf{U}_\bullet A; \Gamma$ yields $\mathcal{C}_1 \circ \mathcal{D}[P_1] \in [\![c : \mathsf{U}_\bullet A]\!]$. I.h. and Lemma D.10 applied to $P_2 \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet A; \Gamma$ yields $\mathcal{C}_2 \circ \mathcal{D}[P_2] \in [\![c : \mathsf{U}_\bullet A]\!]$. By applying Lemma D.8(1) we conclude that $\mathcal{C}_1 \circ \mathcal{C}_2 \circ \mathcal{D}[\mathsf{share}\ c\ \{P_1 \;||\; P_2\}] \in [\![c : \mathsf{U}_\bullet A]\!]$. By Lemma D.10, $\mathsf{share}\ c\ \{P_1 \;||\; P_2\} \in \mathcal{L}[\![\vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\bullet A; \Gamma]\!]$.

Case: [TshL]

$$\frac{P_1 \vdash_\eta \Delta_1, c : \mathsf{U}_\circ A; \Gamma \quad P_2 \vdash_\eta \Delta, c : \mathsf{U}_\bullet A; \Gamma}{\mathsf{share}\ c\ \{P_1 \;||\; P_2\} \;\vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\circ A; \Gamma}$$

Let $\mathcal{C}_1 \in [\![\Delta_1]\!]$, $\mathcal{C}_2 \in [\![\Delta_2]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\begin{aligned}&\mathcal{C}_1 \circ \mathcal{C}_2 \circ \mathcal{D}[\mathsf{share}\ c\ \{P_1 \;||\; P_2\}]\\ &\approx \mathsf{share}\ c\ \{\mathcal{C}_1 \circ \mathcal{D}[P_1] \;||\; \mathcal{C}_2 \circ \mathcal{D}[P_2]\}\end{aligned}$$

I.h. and Lemma D.10 applied to $P_1 \vdash_\eta \Delta_1, c : \mathsf{U}_\circ A; \Gamma$ yields $\mathcal{C}_1 \circ \mathcal{D}[P_1] \in [\![c : \mathsf{U}_\circ A]\!]$.
I.h. and Lemma D.10 applied to $P_2 \vdash_\eta \Delta_2, c : \mathsf{U}_\bullet A; \Gamma$ yields $\mathcal{C}_2 \circ \mathcal{D}[P_2] \in [\![c : \mathsf{U}_\bullet A]\!]$.
By applying Lemma D.8(2) we conclude that $\mathcal{C}_1 \circ \mathcal{C}_2 \circ \mathcal{D}[\mathsf{share}\ c\ \{P_1 \mid\mid P_2\}] \in [\![c : \mathsf{U}_\circ A]\!]$.
By Lemma D.10, $\mathsf{share}\ c\ \{P_1 \mid\mid P_2\} \in \mathcal{L}[\![\vdash_\eta \Delta_1, \Delta_2, c : \mathsf{U}_\circ A; \Gamma]\!]$.

Case: [TshL]. Similarly to case [TshR].

Case: [T**1**]

$$\frac{}{\mathsf{close}\ x \vdash_\eta x : \mathbf{1}; \Gamma}$$

By def. D.3

$$\begin{array}{ll} [\![x : \mathbf{1}]\!] \triangleq S^{\perp\perp}, \text{ where} \\ S \quad\quad = \{Q \vdash x : \mathbf{1} \mid Q \approx \mathsf{close}\ x\}. \end{array}$$

Let $\mathcal{D} \in [\![\Gamma]\!]^!$. We have $\mathcal{D}[\mathsf{close}\ x] \approx \mathsf{close}\ x$. Hence, $\mathcal{D}[\mathsf{close}\ x] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{D}[\mathsf{close}\ x] \in [\![x : \mathbf{1}]\!]$.
Lemma D.10 implies that $\mathsf{close}\ x \in \mathcal{L}[\![x : \mathbf{1}; \Gamma]\!]$.

Case: [T$\otimes$]

$$\frac{P_1 \vdash_\eta \Delta_1, y : A; \Gamma \quad P_2 \vdash_\eta \Delta_2, x : B; \Gamma}{\mathsf{send}\ x(y.P_1); P_2 \ \vdash_\eta \Delta_1, \Delta_2, x : A \otimes B; \Gamma}$$

By def. D.3, $[\![x : A \otimes B]\!] = S^{\perp\perp}$, where

$$S = \{Q \mid \exists Q_1, Q_2.\ Q \approx \mathsf{send}\ x(y.Q_1); Q_2 \text{ and } Q_1 \in [\![y : A]\!]_\sigma \text{ and } Q_2 \in [\![x : B]\!]_\sigma\}.$$

Let $\mathcal{C} \in [\![\Delta_1, \Delta_2]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{send}\ x(y.P_1); P_2] \approx \mathsf{send}\ x(y.\mathcal{C}_1 \circ \mathcal{D}[P_1]); \mathcal{C}_2 \circ \mathcal{D}[P_2]$$

where $\mathcal{C}_1 \in [\![\Delta_1]\!]$ and $\mathcal{C}_2 \in [\![\Delta_2]\!]$.
I.h. and Lemma D.10 applied to $P_1 \vdash_\eta \Delta_1, y : A; \Gamma$ yields $\mathcal{C}_1 \circ \mathcal{D}[P_1] \in [\![y : A]\!]$.
I.h. and Lemma D.10 applied to $P_2 \vdash_\eta \Delta_2, x : B; \Gamma$ yields $\mathcal{C}_2 \circ \mathcal{D}[P_2] \in [\![x : B]\!]$.
Hence, $\mathcal{C} \circ \mathcal{D}[\mathsf{send}\ x(y.P_1); P_2] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{C} \circ \mathcal{D}[\mathsf{send}\ x(y.P_1); P_2] \in [\![x : A \otimes B]\!]$.
Lemma D.10 implies that $\mathsf{send}\ x(y.P_1); P_2 \in \mathcal{L}[\![\vdash_\eta \Delta_1, \Delta_2, x : A \otimes B; \Gamma]\!]$.

Case: [T$\oplus_l$]

$$\frac{P_1 \vdash_\eta \Delta', x : A; \Gamma}{x.\mathsf{inl}; P_1 \ \vdash_\eta \Delta', x : A \oplus B; \Gamma}$$

By def. D.3, $[\![x : A \oplus B]\!] = S^{\perp\perp}$, where

$$S = \{Q \mid \exists Q'.\ (Q \approx x.\mathsf{inl}; Q' \text{ and } Q' \in [\![x : A]\!]_\sigma) \text{ or } (Q \approx x.\mathsf{inr}; Q' \text{ and } Q' \in [\![x : B]\!]_\sigma)\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[x.\mathsf{inl}; P_1] \approx x.\mathsf{inl}; \mathcal{C} \circ \mathcal{D}[P_1]$$

I.h. and Lemma D.10 applied to $P_1 \vdash_\eta \Delta', x : A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P_1] \in [\![x : A]\!]$.
Hence, $\mathcal{C} \circ \mathcal{D}[x.\mathsf{inl}; P_1] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{C} \circ \mathcal{D}[x.\mathsf{inl}; P_1] \in [\![x : A \oplus B]\!]$.
Lemma D.10 implies that $x.\mathsf{inl}; P_1 \in \mathcal{L}[\![\vdash_\eta \Delta', x : A \oplus B; \Gamma]\!]$.

Case: [T$\oplus_r$]. Similarly to case [T$\oplus_l$].

Case: [T!]

$$\frac{P' \vdash_\eta y : A; \Gamma}{!x(y); P' \ \vdash_\eta x : !A; \Gamma}$$

By def. D.3, $[\![x :!A]\!] = S^{\perp\perp}$, where

$$S = \{Q \mid \exists Q'.\ Q \approx !x(y); Q' \text{ and } Q' \in [\![y : A]\!]_\sigma\}.$$

Let $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{D}[!x(y); P'] \approx !x(y); \mathcal{D}[P']$$

I.h. and Lemma D.10 applied to $P' \vdash_\eta y : A; \Gamma$ yields $\mathcal{D}[P'] \in [\![y : A]\!]_\sigma$.
Hence, $\mathcal{D}[P'] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{D}[P'] \in [\![x :!A]\!]_\sigma$.
Lemma D.10 implies that $!x(y); P' \in \mathcal{L}[\![\vdash_\eta x : !A; \Gamma]\!]$.

Case: [T$\exists$]

$$\frac{P' \vdash_\eta \Delta, x : \{B/X\}A; \Gamma}{\mathsf{sendty}\ x(B); P' \ \vdash_\eta \Delta, x : \exists X.A; \Gamma}\ [\mathrm{T}\exists]$$

By def. D.3, $[\![x : \exists X.A]\!] = S^{\perp\perp}$, where

$$S = \{Q \mid \exists Q', S' \ \in \mathcal{R}[- : B].\ Q \approx \mathsf{sendty}\ x(B); Q' \text{ and } Q' \in [\![x : A]\!]_{\sigma[X \mapsto S']}\}.$$

Let $\mathcal{C} \in [\![\Delta]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{sendty}\ x(B); P'] \approx \mathsf{sendty}\ x(B); \mathcal{C} \circ \mathcal{D}[P']$$

I.h. and Lemma D.10 applied to $P' \vdash_\eta \Delta, x : \{B/X\}A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P'] \in [\![x : \{B/X\}A]\!]_\sigma$.
By Lemma D.7(5), $\mathcal{C} \circ \mathcal{D}[P'] \in [\![x : A]\!]_{\sigma[X \mapsto [\![x:B]\!]_\sigma]}$.
Hence, $\mathcal{C} \circ \mathcal{D}[\mathsf{sendty}\ x(B); P'] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{C} \circ \mathcal{D}[\mathsf{sendty}\ x(B); P'] \in [\![x : \exists X.A]\!]$.
Lemma D.10 implies that $\mathsf{sendty}\ x(B); P' \in \mathcal{L}[\![\vdash_\eta \Delta, x : \exists X.A; \Gamma]\!]$.

Case: [Tμ]

$$\frac{P' \vdash_\eta \Delta', x : \{\mu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\mu\ x; P' \vdash_\eta \Delta', x : \mu X.\ A; \Gamma}$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{unfold}_\mu\ x; P'] \approx \mathsf{unfold}_\mu\ x; \mathcal{C} \circ \mathcal{D}[P']$$

I.h. and Lemma D.10 applied to $P' \vdash_\eta \Delta', x : \{\mu X.\ A/X\}A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P'] \in [\![x : \{\mu X.\ A/X\}A]\!]$.
By Lemma D.9(2), $[\![x : \mu X.\ A]\!] = \mathsf{unfold}_\mu\ x; [\![x : \{\mu X.\ A/X\}A]\!]_\sigma$, hence $\mathcal{C} \circ \mathcal{D}[\mathsf{unfold}_\mu\ x; P'] \in [\![x : \mu X.\ A]\!]_\sigma$.
Lemma D.10 implies that $\mathsf{unfold}_\mu\ x; P' \in \mathcal{L}[\![\vdash_\eta \Delta', x : \mu X.\ A; \Gamma]\!]$.

Case: [Tν]

$$\frac{P' \vdash_\eta \Delta', x : \{\mu X.\ A/X\}A; \Gamma}{\mathsf{unfold}_\nu\ x; P' \vdash_\eta \Delta', x : \nu X.\ A; \Gamma}$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{unfold}_\nu\ x; P'] \approx \mathsf{unfold}_\nu\ x; \mathcal{C} \circ \mathcal{D}[P']$$

I.h. and Lemma D.10 applied to $P' \vdash_\eta \Delta', x : \{\mu X.\ A/X\}A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P'] \in [\![x : \{\nu X.\ A/X\}A]\!]$.
By Lemma D.9(5), $\mathsf{unfold}_\nu\ x; [\![x : \{\nu X.\ A/X\}A]\!]_\sigma \subseteq [\![x : \nu X.\ A]\!]_\sigma$, hence $\mathcal{C} \circ \mathcal{D}[\mathsf{unfold}_\nu\ x; P'] \in [\![x : \nu X.\ A]\!]_\sigma$.
Lemma D.10 implies that $\mathsf{unfold}_\nu\ x; P' \in \mathcal{L}[\![\vdash_\eta \Delta', x : \mu X.\ A; \Gamma]\!]$.

Case: [Taffine]

$$\frac{P' \vdash_\eta \vec{c} : \mathsf{U}_\bullet \vec{B}, \vec{a} : \vee\vec{C}, x : A; \Gamma}{\mathsf{affine}\ x; P'\ \vdash_\eta \vec{c} : \mathsf{U}_\bullet \vec{B}, \vec{a} : \vee\vec{C}, a : \wedge A; \Gamma}$$

By def. D.3, $[\![x : \wedge A]\!] = S^{\perp\perp}$, where

$$S = \{Q \mid \exists Q'.\ Q \approx \mathsf{affine}\ x; Q' \text{ and } Q' \in [\![x : A]\!]_\sigma\}.$$

Let $\mathcal{C} \in [\![\vec{c} : \mathsf{U}_\bullet \vec{B}, \vec{a} : \vee\vec{C}]\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$. We have

$$\mathcal{C} \circ \mathcal{D}[\mathsf{affine}\ x; P'] \approx \mathsf{affine}\ x; \mathcal{C} \circ \mathcal{D}[P']$$

I.h. and Lemma D.10 applied to $P' \vdash_\eta \vec{c} : \mathsf{U}_\bullet \vec{B}, \vec{a} : \vee\vec{C}, x : A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P'] \in [\![x : A]\!]$.
Hence, $\mathcal{C} \circ \mathcal{D}[\mathsf{affine}\ x; P'] \in S$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, thus $\mathcal{C} \circ \mathcal{D}[\mathsf{affine}\ x; P'] \in [\![x : \wedge A]\!]$.
Lemma D.10 implies that $\mathsf{affine}\ x; P' \in \mathcal{L}[\![\vdash_\eta \vec{c} : \mathsf{U}_\bullet \vec{B}, \vec{a} : \vee\vec{C}, x : A; \Gamma]\!]$.

Case: [Tcell]

$$\frac{P' \vdash_\eta \Delta', a : \wedge A; \Gamma}{\mathsf{cell}\ c(a.P') \vdash_\eta \Delta', c : \mathsf{S}_\bullet A; \Gamma}$$

Let $\mathcal{C} \in [\![\Delta']\!]$, $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in [\![c : \mathsf{U}_\bullet \overline{A}]\!]$.
I.h. and Lemma D.10 applied to $P' \vdash_\eta \Delta', a : \wedge A; \Gamma$ yields $\mathcal{C} \circ \mathcal{D}[P'] \in [\![a : \wedge A]\!]$.
Since $Q \in [\![c : \mathsf{U}_\bullet \overline{A}]\!]$, then Q is $[\![a : \wedge A]\!]$-preserving.
Hence, by Lemma D.2(1), $\mathsf{cell}\ c(a.\mathcal{C} \circ \mathcal{D}[P'])\ |c|\ Q$ is simulated by $\mathsf{cell}\ c(a.[\![a : \wedge A]\!])\ |c|\ Q$.
Since $Q \in [\![c : \mathsf{U}_\bullet \overline{A}]\!] = S^\perp$ where $S = \{R \mid R \approx \mathsf{cell}\ c(a.[\![a : \wedge A]\!])\}$, then $\mathsf{cell}\ c(a.[\![a : \wedge A]\!])\ |c|\ Q$ is SN.
Hence, $\mathcal{C} \circ \mathcal{D}[\mathsf{cell}\ c(a.P')]\ |c|\ Q$ is SN.
Then, $\mathsf{cell}\ c(a.P') \in \mathcal{L}[\![\vdash_\eta \Delta', c : \mathsf{S}_\bullet A; \Gamma]\!]$.

Case: [Tempty]

$$\frac{}{\mathsf{empty}\ c \vdash_\eta c : \mathsf{S}_\circ A; \Gamma}$$

Let $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in [\![c : \mathsf{U}_\circ \overline{A}]\!]$.
Since $Q \in [\![c : \mathsf{U}_\circ \overline{A}]\!]$, then Q is $[\![a : \wedge A]\!]$-preserving.
Hence, by Lemma D.2(2), $\mathsf{empty}\ c\ |c|\ Q$ is simulated by $\mathsf{empty}\ c([\![a : \wedge A]\!].\ )|c|\ Q$.
Since $Q \in [\![c : \mathsf{U}_\circ \overline{A}]\!] = S^\perp$ where $S = \{R \mid R \approx \mathsf{empty}\ c([\![a : \wedge A]\!].\})$, then $\mathsf{empty}\ c([\![a : \wedge A]\!].\ )|c|\ Q$ is SN.
Hence, $\mathcal{D}[\mathsf{empty}\ c]\ |c|\ Q$ is SN.
Then, $\mathsf{empty}\ c \in \mathcal{L}[\![\vdash_\eta c : \mathsf{S}_\circ A; \Gamma]\!]$.

Case: [T⊥]

$$\frac{P' \vdash_\eta \Delta'; \Gamma}{\mathsf{wait}\ x; P'\ \vdash_\eta \Delta', x : \bot; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : \bot]\!] = S^\perp$, where

$$S = \{Q \vdash x : \mathbf{1} \mid Q \approx \mathsf{close}\ x\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then, $Q \approx \mathsf{close}\ x$.
We prove that (H) $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P']$ is SN, by induction on $N(Q)+N(\mathcal{C})$.
Suppose that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P'] \approx \mathsf{close}\ x\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P'] \to \mathcal{C} \circ \mathcal{D}[P']\ = R$$

Applying i.h. to $P' \vdash_\eta \Delta'; \Gamma$ yields R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P']$ is SN.
Therefore, $\mathcal{C} \circ \mathcal{D}[\mathsf{wait}\ x; P']\ \in [\![x : \bot]\!]_\sigma$.
By Lemma D.10, $\mathsf{wait}\ x; P' \in \mathcal{L}[\![\vdash_\eta \Delta', x : \bot; \Gamma]\!]$.

Case: [T⅋]

$$\frac{P' \vdash_\eta \Delta', z : A, x : B; \Gamma}{\mathsf{recv}\ x(z); P' \vdash_\eta \Delta', x : A \parr B; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : A \parr B]\!] = S^\perp$, where

$$S = \{Q \mid \exists Q_1, Q_2.\ Q \approx \mathsf{send}\ x(y.Q_1); Q_2 \text{ and } Q_1 \in [\![y : \overline{A}]\!]_\sigma \text{ and } Q_2 \in [\![x : \overline{B}]\!]_\sigma\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then, $Q \approx \mathsf{send}\ x(y.Q_1); Q_2$ and $Q_1 \in [\![y : \overline{A}]\!]_\sigma$ and $Q_2 \in [\![x : \overline{B}]\!]_\sigma$.
We prove that (H) $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P']$ is SN, by induction on $N(Q) + N(\mathcal{C})$.
Suppose that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P'] \to R$. There are two cases to consider:

Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$\begin{aligned} Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P'] &\approx \mathsf{send}\ x(y.Q_1); Q_2 \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P'] \\ &\to Q_2 \;|x|\; (Q_1 \;|y|\; \mathcal{C} \circ \mathcal{D}[\{y/z\}P']) \;= R \end{aligned}$$

Applying i.h. to $\{y/z\}P' \vdash_\eta \Delta', y : A, x : B; \Gamma$ yields R is SN. In either case (i)-(ii), R is SN. By applying Lemma D.5(3) we conclude that $Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P']$ is SN. Therefore, $\mathcal{C} \circ \mathcal{D}[\mathsf{recv}\ x(z); P'] \in [\![x : A \parr B]\!]_\sigma$. By Lemma D.10, $\mathsf{recv}\ x(z); P' \in \mathcal{L}[\![\vdash_\eta \Delta', x : A \parr B; \Gamma]\!]$.

Case: [T&]

$$\frac{P_1 \vdash_\eta \Delta', x : A; \Gamma \quad P_2 \vdash_\eta \Delta', x : B; \Gamma}{\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}\ \vdash_\eta \Delta', x : A \,\&\, B; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : A \,\&\, B]\!] = S^\perp$, where

$$S = \{Q \mid \exists Q'.\, (Q \approx x.\mathsf{inl}; Q' \text{ and } \ Q' \in [\![x : \overline{A}]\!]_\sigma) \ \text{ or } \ (Q \approx x.\mathsf{inr}; Q' \text{ and } Q' \in [\![x : \overline{B}]\!]_\sigma)\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$. Suppose that $Q \approx x.\mathsf{inl}; Q'$ and $Q' \in [\![x : \overline{A}]\!]_\sigma$. The case in which choice is right is handled similarly. We prove that (H) $Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}]$ is SN, by induction on $N(Q) + N(\mathcal{C})$. Suppose that $Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii).

$$\begin{aligned} &Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}] \\ &\approx x.\mathsf{inl}; Q_1 \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}] \\ &\to Q_1 \;|x|\; \mathcal{C} \circ \mathcal{D}[P_1] = R \end{aligned}$$

Applying i.h. to $P_1 \vdash_\eta \Delta', x : A; \Gamma$ yields R is SN. In either case (i)-(ii), R is SN. By applying Lemma D.5(3) we conclude that $Q \;|x|\; \mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}]$ is SN. Therefore, $\mathcal{C} \circ \mathcal{D}[\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\}] \in [\![x : A \,\&\, B]\!]_\sigma$. By Lemma D.10, $\mathsf{case}\ x\ \{|\mathsf{inl} : P_1,\ |\mathsf{inr} : P_2\} \in \mathcal{L}[\![\vdash_\eta \Delta', x : A \,\&\, B; \Gamma]\!]$.

Case: [T?]

$$\frac{P' \vdash_\eta \Delta'; \Gamma, x : A}{?x; P'\ \vdash_\eta \Delta', x : ?A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : ?A]\!] = S^{\perp}$, where

$$S = \{Q \mid \exists Q'.\ Q \approx {!}x(y); Q' \text{ and } Q' \in [\![y : \overline{A}]\!]_{\sigma}\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^{!}$ and $Q \in S$.
Then, $Q \approx {!}x(y); Q'$ and $Q' \in [\![y : \overline{A}]\!]_{\sigma}$.
We prove that (H) $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[?x; P']$ is SN, by induction on $N(Q) + N(\mathcal{C})$.
Suppose that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[?x; P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$\begin{aligned}
& Q\ |x|\ \mathcal{C} \circ \mathcal{D}[?x; P'] \\
& \approx {!}x(y); Q'\ |x|\ \mathcal{C} \circ \mathcal{D}[?x; P'] \\
& \to y.Q'\ |{!}x|\ \mathcal{C} \circ \mathcal{D}[P'] = R
\end{aligned}$$

Applying i.h. to $P' \vdash_{\eta} \Delta'; \Gamma, x : A$ yields R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[?x; P']$ is SN.
Therefore, $\mathcal{C} \circ \mathcal{D}[?x; P'] \in [\![x : ?A]\!]_{\sigma}$.
By Lemma D.10, $?x; P' \in \mathcal{L}[\![\vdash_{\eta} \Delta', x : ?A; \Gamma]\!]$.

Case: [Tcall]

$$\frac{P' \vdash_{\eta} \Delta, z : A; \Gamma', x : A}{\mathsf{call}\ x(z); P' \ \vdash_{\eta} \Delta; \Gamma', x : A}$$

Let $\mathcal{C} \in [\![\Delta]\!]$ and $\mathcal{D} \in [\![\Gamma', x : A]\!]^{!}$. We prove that (H) $\mathcal{C} \circ \mathcal{D}[\mathsf{call}\ x(z); P']$ is SN, by induction on $N(\mathcal{C})$.
Suppose that $\mathcal{C} \circ \mathcal{D}[\mathsf{call}\ x(z); P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of$\mathcal{C}$.
Case: (ii) R is obtained by an interaction on session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$\begin{aligned}
& \mathcal{C} \circ \mathcal{D}[\mathsf{call}\ x(z); P'] \\
& \approx y.Q\ |{!}x|\ \mathcal{C} \circ \mathcal{D}'[\mathsf{call}\ x(z); P'] \\
& \to (\{z/y\}Q\ |z|\ \mathcal{C}) \circ (y.Q\ |{!}x|\ \mathcal{D}')[P'] = R
\end{aligned}$$

Since $\mathcal{D} \in [\![\Gamma', x : A]\!]^{!}$, then $\mathcal{D}' \in [\![\Gamma']\!]^{!}$ and $Q \in [\![y : \overline{A}]\!]$ (Def. D.7).
By Lemma D.7(1), $\{z/y\}Q \in [\![z : \overline{A}]\!]$.
Then, $\{z/y\}Q\ |z|\ \mathcal{C} \in [\![\Delta, z : A]\!]$ and $y.Q\ |{!}x|\ \mathcal{D}' \in [\![\Gamma', x : A]\!]^{!}$ (Def. D.7).
Applying i.h. to $P' \vdash_{\eta} \Delta, z : A; \Gamma', x : A$ yields R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $\mathcal{C} \circ \mathcal{D}[\mathsf{call}\ x(z); P']$ is SN.
Thus, $\mathsf{call}\ x(z); P' \ \in \mathcal{L}[\![\vdash_{\eta} \Delta; \Gamma', x : A]\!]$.

Case: [T∀]

$$\frac{P' \vdash_\eta \Delta', x : A; \Gamma}{\mathsf{recvty}\ x(X); P' \vdash_\eta \Delta', x : \forall X.A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : \forall X.A]\!] = S^\perp$, where

$$S = \{Q \mid \exists Q', S' \in \mathcal{R}[- : B].\ Q \approx \mathsf{sendty}\ x(B); Q' \text{ and } Q' \in [\![x : \overline{A}]\!]_{\sigma[X \mapsto S']}\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$. Then, $Q \approx \mathsf{sendty}\ x(B); Q'$ and $Q' \in [\![x : \overline{A}]\!]_{\sigma[X \mapsto S']}$, for some $S' \in \mathcal{R}[- : B]$. We prove that (H) $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P']$ is SN, by induction on $N(Q) + N(\mathcal{C})$. Suppose that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$\begin{aligned} Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P'] &\approx \mathsf{sendty}\ x(B); Q'\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P'] \\ &\to Q'\ |x|\ \mathcal{C} \circ \mathcal{D}[\{B/X\}P'] \ = R \end{aligned}$$

Applying i.h. to $\{B/X\}P' \vdash_\eta \Delta', x : \{B/X\}A; \Gamma$ and Lemma D.10 yields $\mathcal{C} \circ \mathcal{D}[\{B/X\}P'] \in [\![x : \{B/X\}A]\!]_\sigma$.
By Lemma D.7(5), $\mathcal{C} \circ \mathcal{D}[\{B/X\}P'] \in [\![x : A]\!]_{\sigma[X \mapsto S']}$.
Since $Q' \in [\![x : \overline{A}]\!]_{\sigma[X \mapsto S']}$ and $\mathcal{C} \circ \mathcal{D}[\{B/X\}P'] \in [\![x : A]\!]_{\sigma[X \mapsto S']}$, Lemma D.7(4) yields that R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |x|\ \mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P']$ is SN.
Therefore, $\mathcal{C} \circ \mathcal{D}[\mathsf{recvty}\ x(X); P'] \in [\![x : \forall X.A]\!]_\sigma$.
By Lemma D.10, $\mathsf{recvty}\ x(X); P' \in \mathcal{L}[\![\vdash_\eta \Delta', x : \forall X.A; \Gamma]\!]$.

Case: [Tcorec]

$$\frac{\{x/z\}\{\vec{y}/\vec{w}\}P' \vdash_{\eta'} \Delta', x : A; \Gamma \quad \eta' = \eta, Y(x, \vec{y}) \mapsto \Delta', x : X; \Gamma}{\mathsf{corec}\ Y(z, \vec{w}); P'\ [x, \vec{y}] \vdash_\eta \Delta', x : \nu X.\ A; \Gamma}$$

Let $\rho \in [\![\eta]\!]_\sigma$, $\mathcal{C} \in [\![\Delta']\!]_\sigma$ and $\mathcal{D} \in [\![\Gamma]\!]^!{}_\sigma$.
We prove that $\mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z, \vec{w}); P'\ [x, \vec{y}])] \in [\![x : \nu X.\ A]\!]_\sigma$.
By Lemma D.10, this implies that $\mathsf{corec}\ Y(z, \vec{w}); P'\ [x, \vec{y}] \in \mathcal{L}[\![\vdash_\eta \Delta', x : \nu X.\ A; \Gamma]\!]_\sigma$.
By Lemma D.9(5), we have

$$[\![x : \nu X.\ A]\!]_\sigma = \bigcap_{n \in \mathbb{N}} \phi^n_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp$$

where $\phi_{\{\overline{X}/X\}\overline{A}}(S) \triangleq \mathsf{unfold}_\mu\ x; [\![x : \{\overline{X}/X\}\overline{A}]\!]_{\sigma[X \mapsto S]}$.

We prove (H1):

$$\forall n \in \mathbb{N},\ \forall \rho \in [\![\eta]\!]_\sigma,\ \forall \mathcal{C} \in [\![\Delta']\!]_\sigma,\ \forall \mathcal{D} \in [\![\Gamma]\!]^!{}_\sigma.\\ \mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])] \in \phi^n_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp$$

Proof of (H1) is by induction on $n \in \mathbb{N}$:
Case: $n = 0$.
Follows because $\mathcal{C}\circ\mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])] \in \emptyset^\perp$ and since $\phi^0_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp = \emptyset^{\perp\perp\perp} = \emptyset^\perp$ (Lemma D.6(5)).
Case: $n = m + 1$.
Let $Q \in \phi^{m+1}_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})$.
Then $Q \approx \mathsf{unfold}_\mu\ x; Q'$, where $Q' \in [\![x : \{\overline{X}/X\}\overline{A}]\!]_{\sigma[X \mapsto \psi^m_A(\emptyset^{\perp\perp})]}$.
We prove (H2)

$$\mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])]\ |x|\ Q \text{ is SN}$$

by induction on $N(\mathcal{C}) + N(\rho) + N(Q)$.
Suppose that $\mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])]\ |x|\ Q \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either $\mathcal{C}$, ρ or Q.
Case: (ii) R is obtained by an interaction on session x.
Case (i) follows by inner inductive hypothesis (H2).
So let us consider case (ii). Then

$$\begin{aligned}&\mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])]\ |x|\ Q\\ &\approx \mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])]\ |x|\ \mathsf{unfold}_\mu\ x; Q'\\ &\to \mathcal{C} \circ \mathcal{D}[\rho(\{x/z\}\{\vec{y}/\vec{w}\}\{\mathsf{corec}\ Y(z,\vec{w}); P'/Y\}P')]\ |x|\ Q'\\ &= \mathcal{C} \circ \mathcal{D}[\rho'(\{x/z\}\{\vec{y}/\vec{w}\}P')]\ |x|\ Q' = R\end{aligned}$$

where $\rho' = \rho, Y(x,\vec{y}) \mapsto \rho(\mathsf{corec}\ Y(z,\vec{w}); P')$.
I.h. (H1) applied to m yields

$$\forall \mathcal{C} \in [\![\Delta']\!],\ \forall \mathcal{D} \in [\![\Gamma]\!]^!.\\ \mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])] \in \phi^m_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp$$

Hence, by Lemma D.10, we obtain

$$\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}]) \in \mathcal{L}[\![\vdash_\emptyset \Delta', x : X; \Gamma]\!]_{\sigma[X \mapsto \psi^m_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp]}$$

Therefore, $\rho' \in [\![\eta']\!]_\sigma$.
Applying i.h. (outer i.h., fundamental lemma) to $\{x/z\}\{\vec{y}/\vec{w}\}P' \vdash_{\eta'} \Delta', x : A; \Gamma$ and Lemma D.10 yields $\mathcal{C} \circ \mathcal{D}[\rho'(\{x/z\}\{\vec{y}/\vec{w}\}P')] \in [\![x : A]\!]_{\sigma[X \mapsto \psi^m_A(\emptyset^{\perp\perp})^\perp]}$.
Lemma D.7(6) implies $\mathcal{C}\circ\mathcal{D}[\rho'(\{x/z\}\{\vec{y}/\vec{w}\}P')] \in [\![x : \{\overline{X}/X\}A]\!]_{\sigma[X \mapsto \psi^m_A(\emptyset^{\perp\perp})]}$.
By hypothesis, $Q' \in [\![x : \{\overline{X}/X\}\overline{A}]\!]_{\sigma[X \mapsto \psi^m_A(\emptyset^{\perp\perp})]}$, hence by Lemma D.7(3) we obtain that R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $\mathcal{C}\circ\mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])]\ |x|\ Q$ is SN.
Therefore, $\mathcal{C} \circ \mathcal{D}[\rho(\mathsf{corec}\ Y(z,\vec{w}); P'\ [x,\vec{y}])] \in \phi^{m+1}_{\{\overline{X}/X\}\overline{A}}(\emptyset^{\perp\perp})^\perp$.

Case: [Tdiscard]

$$\frac{}{\mathsf{discard}\ a \vdash_\eta a : \vee A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : \vee A]\!] = S^\perp$, where

$$S = \{Q \mid \exists Q'.\ Q \approx \mathsf{affine}\ a; Q' \text{ and } Q' \in [\![a : \overline{A}]\!]_\sigma\}.$$

Let $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then, $Q \approx \mathsf{affine}\ a; Q'$ and $Q' \in [\![a : \overline{A}]\!]_\sigma$.
We have $Q\ |a|\ \mathcal{D}[\mathsf{discard}\ a] \approx Q\ |a|\ \mathsf{discard}\ a$.
We prove that (H) $Q\ |a|\ \mathsf{discard}\ a$ is SN, by induction on $N(Q)$.
Suppose that $Q\ |a|\ \mathsf{discard}\ a \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q.
Case: (ii) R is obtained by an interaction on cut session a.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$Q\ |a|\ \mathsf{discard}\ a \approx \mathsf{affine}\ a; Q'\ |a|\ \mathsf{discard}\ a \to \mathsf{0} = R$$

In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |a|\ \mathsf{discard}\ a$ is SN.
Therefore, $\mathsf{discard}\ a\ \in [\![x : \vee A]\!]_\sigma$, hence $\mathcal{D}[\mathsf{discard}\ a] \in [\![x : \vee A]\!]_\sigma$ (Lemma D.7(2)).
By Lemma D.10, $\mathsf{discard}\ a \in \mathcal{L}[\![\vdash_\eta a : \vee A; \Gamma]\!]$.

Case: [Tuse]

$$\frac{P' \vdash_\eta \Delta', a : A; \Gamma}{\mathsf{use}\ a; P' \vdash_\eta \Delta', a : \vee A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : \bot]\!] = S^\perp$, where

$$S = \{Q \mid \exists Q'.\ Q \approx \mathsf{affine}\ a; Q' \text{ and } Q' \in [\![a : \overline{A}]\!]_\sigma\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then $Q \approx \mathsf{affine}\ a; Q'$, where $Q' \in [\![a : \overline{A}]\!]$.
We prove that (H) $Q\ |a|\ \mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P']$ is SN, by induction on $N(Q)+N(\mathcal{C})$.
Suppose that $Q\ |a|\ \mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session x.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$\begin{aligned} Q\ |a|\ \mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P'] &\approx \mathsf{affine}\ a; Q'\ |a|\ \mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P'] \\ &\to (Q'\ |a|\ \mathcal{C}) \circ \mathcal{D}[P']\ = R \end{aligned}$$

Applying i.h. to $P' \vdash_\eta \Delta', a : A; \Gamma$ yields R is SN.
In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |a|\ \mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P']$ is SN.
Therefore, $\mathcal{C} \circ \mathcal{D}[\mathsf{use}\ a; P']\ \in [\![a : \vee A]\!]_\sigma$.
By Lemma D.10, $\mathsf{use}\ a; P' \in \mathcal{L}[\![\vdash_\eta \Delta', a : A; \Gamma]\!]$.

Case: [Trelease]

$$\frac{}{\mathsf{release}\ c \vdash_\eta c : \mathsf{U}_\bullet A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![x : \mathsf{U}_\bullet A]\!] = S^\perp$, where

$$S = \{Q \mid Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\}.$$

Let $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then, $Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma$.
We prove that (H) $Q\ |c|\ \mathcal{D}[\mathsf{release}\ c]$ is SN, by induction on $N(Q)$.
Suppose that $Q\ |c|\ \mathcal{D}[\mathsf{release}\ c] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q.
Case: (ii) R is obtained by an interaction on cut session c.
Case (i) follows by inner inductive hypothesis (H).
So let us consider case (ii). Then

$$Q\ |c|\ \mathcal{D}[\mathsf{release}\ c] \approx \mathcal{D}[\mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\ |c|\ \mathsf{release}\ c] \xrightarrow{*}_{\mathsf{c}} \mathcal{D}[\mathbf{0}] = R$$

In either case (i)-(ii), R is SN.
By applying Lemma D.5(3) we conclude that $Q\ |c|\ \mathcal{D}[\mathsf{release}\ c]$ is SN.
Furthermore, $\mathsf{release}\ c$ is vacuously $[\![y : \wedge\overline{A}]\!]_\sigma$-preserving, for any y.
Therefore, $\mathcal{D}[\mathsf{release}\ c] \in [\![x : \mathsf{U}_\bullet A]\!]_\sigma$.
By Lemma D.10, $\mathsf{release}\ c \in \mathcal{L}[\![\vdash_\eta a : \mathsf{U}_\bullet A; \Gamma]\!]$.

Case: [Ttake]

$$\frac{P' \vdash_\eta \Delta', a : \vee A, c : \mathsf{U}_\circ A; \Gamma}{\mathsf{take}\ c(a); P' \vdash_\eta \Delta', c : \mathsf{U}_\bullet A; \Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![c : \mathsf{U}_\bullet A]\!] = S^\perp$, where

$$S = \{Q \mid Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\}.$$

Let $\mathcal{C} \in [\![\Delta']\!]$ and $\mathcal{D} \in [\![\Gamma]\!]^!$ and $Q \in S$.
Then, $Q \approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma$.
We prove that (H) $Q\ |c|\ \mathcal{C} \circ \mathcal{D}[\mathsf{take}\ c(a); P']$ is SN, by induction on $N(Q) + N(\mathcal{C})$.
Suppose that $Q\ |c|\ \mathcal{C} \circ \mathcal{D}[\mathsf{take}\ c(a); P'] \to R$. There are two cases to consider:
Case: (i) R is obtained by an internal reduction of either Q or $\mathcal{C}$.
Case: (ii) R is obtained by an interaction on cut session c.
Case (i) follows by inner inductive hypothesis (H). So let us consider case (ii). Then

$$\begin{aligned} Q\ |c|\ \mathcal{C} \circ \mathcal{D}[\mathsf{take}\ c(a); P'] &\approx \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\ |c|\ \mathcal{C} \circ \mathcal{D}[\mathsf{take}\ c(a); P'] \\ &\to \mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma\ |c|\ (Q'\ |a|\ \mathcal{C} \circ \mathcal{D}[P'])\ = R \end{aligned}$$

where $Q' \in [\![a : \wedge\overline{A}]\!]_\sigma$.
By Def. D.3, $[\![c : \mathsf{S}_\bullet \overline{A}]\!] = S^{\perp\perp}$.
By Lemma D.6(4), $S \subseteq S^{\perp\perp}$, hence $\mathsf{cell}\ c(a.[\![a : \wedge\overline{A}]\!])_\sigma \in [\![c : \mathsf{S}_\bullet \overline{A}]\!]$.
Applying i.h. to $P' \vdash_\eta \Delta', a : \vee A, c : \mathsf{U}_\circ A; \Gamma$ yields R is SN.

In either case (i)-(ii), R is SN. By applying Lemma D.5(3) we conclude that $Q\ |c|\ \mathcal{C}\circ\mathcal{D}[\mathsf{take}\ c(a);P']$ is SN. Now, we prove that $\mathcal{C}\circ\mathcal{D}[\mathsf{take}\ c(a);P']$ is $[\![a:\wedge\overline{A}]\!]_\sigma$-preserving, for any a. Let $R\in[\![a:\wedge\overline{A}]\!]_\sigma$. Applying i.h. to $P'\vdash_\eta \Delta', a:\vee A, c:\mathsf{U}_\circ A;\Gamma$ we conclude that $R\ |a|\ \mathcal{C}\circ\mathcal{D}[P']\in[\![c:\mathsf{U}_\circ A]\!]_\sigma$, which implies that $R\ |a|\ \mathcal{C}\circ\mathcal{D}[P']\in[\![c:\mathsf{U}_\circ A]\!]_\sigma$ and hence $R\ |a|\ \mathcal{C}\circ\mathcal{D}[P']$ is $[\![a:\wedge\overline{A}]\!]_\sigma$-preserving. Therefore, $\mathcal{C}\circ\mathcal{D}[\mathsf{take}\ c(a);P']\in[\![c:\mathsf{U}_\bullet A]\!]_\sigma$. By Lemma D.10, $\mathsf{take}\ c(a);P'\in\mathcal{L}[\![\vdash_\eta \Delta', c:\mathsf{U}_\bullet A;\Gamma]\!]$.

Case: [Tput]

$$\frac{P_1\vdash_\eta \Delta_1, a:\wedge\overline{A};\Gamma \qquad P_2\vdash_\eta \Delta_2, c:\mathsf{U}_\bullet A;\Gamma}{\mathsf{put}\ c(a.P_1);P_2\ \vdash_\eta \Delta_1,\Delta_2, c:\mathsf{U}_\circ A;\Gamma}$$

By Def. D.3 and Lemma D.6(5) we have $[\![c:\mathsf{U}_\circ A]\!]=S^\perp$, where

$$S=\{Q\mid Q\approx \mathsf{empty}\ c([\![a:\wedge\overline{A}]\!]_\sigma.\}).$$

Let $\mathcal{C}_1\in[\![\Delta_1]\!],\mathcal{C}_2\in[\![\Delta_2]\!]$ and $\mathcal{D}\in[\![\Gamma]\!]^!$ and $Q\in S$. Then, $Q\approx\mathsf{empty}\ c([\![a:\wedge\overline{A}]\!].)\sigma$. We prove that (H) $Q\ |c|\ \mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]$ is SN, by induction on $N(Q)+N(\mathcal{C}_1)+N(\mathcal{C}_2)$. Suppose that $Q\ |c|\ \mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]\to R$. There are two cases to consider:

Case: (i) R is obtained by an internal reduction of either Q, $\mathcal{C}_1$ or $\mathcal{C}_2$.

Case: (ii) R is obtained by an interaction on cut session c.

Case (i) follows by inner inductive hypothesis (H).So let us consider case (ii). Then

$$\begin{aligned}&Q\ |c|\ \mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]\approx\mathsf{empty}\ c([\![a:\wedge A]\!]_\sigma.\ )|c|\ \mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]\\&\approx\mathsf{empty}\ c([\![a:\wedge A]\!]_\sigma.\ )|c|\ \mathsf{put}\ c(a.\mathcal{C}_1\circ\mathcal{D}[P_1]);\mathcal{C}_2\circ\mathcal{D}[P_2]\\&\to\mathsf{cell}\ c(a.[\![a:\wedge\overline{A}]\!])_\sigma\ |c|\ \mathcal{C}_2\circ\mathcal{D}[P_2]\ =R\ (*)\end{aligned}$$

I.h. applied to $P_1\vdash_\eta \Delta_1, a:\wedge\overline{A};\Gamma$ yields $\mathcal{C}_1\circ\mathcal{D}[P_1]\in[\![a:\wedge\overline{A}]\!]$, hence reduction step (*). By Def. D.3, $[\![c:\mathsf{S}_\bullet\overline{A}]\!]=S^{\perp\perp}$. By Lemma D.6(4), $S\subseteq S^{\perp\perp}$, hence $\mathsf{cell}\ c(a.[\![a:\wedge\overline{A}]\!])_\sigma\in[\![c:\mathsf{S}_\bullet\overline{A}]\!]$. Applying i.h. to $P_2\vdash_\eta \Delta_2, c:\mathsf{U}_\bullet A;\Gamma$ yields R is SN. In either case (i)-(ii), R is SN. By applying Lemma D.5(3) we conclude that $Q\ |c|\ \mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]$ is SN. Now, we prove that $\mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]$ is $[\![a:\wedge\overline{A}]\!]_\sigma$-preserving, for any a. Applying i.h. to $P_1\vdash_\eta \Delta_1, a:\wedge\overline{A};\Gamma$ we conclude that $\mathcal{C}_1\circ\mathcal{D}[P_1]\in[\![a:\wedge\overline{A}]\!]$. Applying i.h. to $P_2\vdash_\eta \Delta_2, c:\mathsf{U}_\bullet A;\Gamma$ we conclude that $\mathcal{C}_2\circ\mathcal{D}[P_2]\in[\![c:\mathsf{U}_\bullet A]\!]$, which implies that $\mathcal{C}_2\circ\mathcal{D}[P_2]$ is $[\![a:\wedge\overline{A}]\!]_\sigma$-preserving Therefore, $\mathcal{C}_1\circ\mathcal{C}_2\circ\mathcal{D}[\mathsf{put}\ c(a.P_1);P_2]\in[\![c:\mathsf{U}_\circ A]\!]_\sigma$. By Lemma D.10, $\mathsf{put}\ c(a.P_1);P_2\in\mathcal{L}[\![\vdash_\eta \Delta_1,\Delta_2, c:\mathsf{U}_\circ A;\Gamma]\!]$.

Theorem D.1 (Strong Normalisation). *If $P\vdash_\emptyset \emptyset;\emptyset$, then P is SN.*

Proof. Immediately by Lemma D.11.