



Linear logical relations and observational equivalences for session-based concurrency

Jorge A. Pérez^{a,*}, Luís Caires^b, Frank Pfenning^c, Bernardo Toninho^{b,c}

^a Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Netherlands

^b CITI, NOVA LINES and Departamento de Informática, FCT Universidade Nova de Lisboa, Portugal

^c Computer Science Department, Carnegie Mellon University, United States

ARTICLE INFO

Article history:

Received 12 September 2013

Received in revised form 22 April 2014

Available online 11 August 2014

Keywords:

Session types

Linear logic

Process calculi

Strong normalization

Confluence

Logical relations

Observational equivalences

ABSTRACT

We investigate *strong normalization*, *confluence*, and *behavioral equality* in the realm of session-based concurrency. These interrelated issues underpin advanced correctness analysis in models of structured communications. The starting point for our study is an interpretation of linear logic propositions as session types for communicating processes, proposed in prior work. Strong normalization and confluence are established by developing a theory of *logical relations*. Defined upon a linear type structure, our logical relations remain remarkably similar to those for functional languages. We also introduce a natural notion of *observational equivalence* for session-typed processes. Strong normalization and confluence come in handy in the associated coinductive reasoning: as applications, we prove that all *proof conversions* induced by the logic interpretation actually express observational equivalences, and explain how *type isomorphisms* resulting from linear logic equivalences are realized by coercions between interface types of session-based concurrent systems.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Modern computing systems rely heavily on the *communication* between distributed software artifacts. Hence, to a large extent, guaranteeing system correctness amounts to ensuring consistent dialogues between such artifacts. This is a challenging task, given the complex interaction patterns that communicating systems usually feature. *Session-based concurrency* provides a foundational approach to communication correctness: concurrent dialogues are structured into basic units called *sessions*; descriptions of the interaction patterns are then abstracted as *session types* [22,23,17], which are statically checked against specifications. These specifications are usually given in the π -calculus [31,42], and so we obtain *processes* interacting on so-called session channels which connect exactly two subsystems. The discipline of session types ensures protocols in which actions always occur in dual pairs: when one partner sends, the other receives; when one partner offers a selection, the other chooses; when a session terminates, no further interaction may occur. New sessions may be dynamically created by invoking shared servers. While *concurrency* arises in the simultaneous execution of sessions, *mobility* results from the exchange of session and server names.

The goal of this paper is to investigate *strong normalization*, *confluence*, and *typed behavioral equivalences* for session-typed, communicating processes. These interrelated issues underpin advanced correctness analysis in models of structured

* Corresponding author.

E-mail addresses: j.a.perez@rug.nl (J.A. Pérez), lcaires@fct.unl.pt (L. Caires), fp@cs.cmu.edu (F. Pfenning), btoninho@cs.cmu.edu (B. Toninho).

communications. Our study builds upon the interpretation of linear logic propositions as session types put forward by Caires and Pfenning in [11]. In a concurrent setting, such an interpretation defines a tight propositions-as-types/proofs-as-programs correspondence, in the style of the Curry–Howard isomorphism for the simply-typed λ -calculus [24]. In the interpretation, types (linear logic propositions) are assigned to names (communication channels) and describe their session protocol; typing rules correspond to linear sequent calculus proof rules and processes correspond to proof objects in logic derivations. Moreover, process reduction may be simulated by proof conversions and reductions, and vice versa. As a result, typed processes enjoy strong forms of *subject reduction* (type preservation) and *global progress* (deadlock-freedom). While the former states that well-typed processes always evolve to well-typed processes (a *safety* property), the latter says that well-typed processes will never get into a stuck state (a *liveness* property). These strong correctness properties make the framework in [11] a convenient basis for our study of strong normalization, confluence, and behavioral equivalences. Well-studied in sequential settings, these three interrelated issues constitute substantial challenges for theories of communicating processes, as we motivate next.

In typed functional calculi, *strong normalization* ensures that well-typed terms do not have infinite reduction sequences. Types rule out divergent computations; termination of reduction entails consistency of the corresponding logical systems. In the realm of communicating processes, reduction captures atomic synchronization; associated behavioral types exclude unintended structured interactions. As a result, strong normalization acquires an enhanced significance in a concurrent setting. In fact, even if subject reduction and progress are typically regarded as the key correctness guarantees for processes, requiring strongly normalizing behaviors is also most sensible: while from a global perspective systems are meant to run forever, at a local level we wish responsive participants which always react within a finite amount of time, and never engage into infinite internal behavior. For instance, in server-client applications it is critical for clients to be sure that running code provided by the server will not cause them to get stuck indefinitely (as in a denial-of-service attack, or just due to some bug).

Closely related to strong normalization, *confluence* is another appealing property. In a communication-based setting, confluence would strengthen correctness guarantees by ensuring *predictability* of structured interactions. This benefit may be more concretely seen by considering the principle of *typed process composition* derived from the logic interpretation. In [11], typing judgments specify both the session behavior that a process *offers* (or *implements*) and the set of (unrestricted and linear) behaviors that it *requires* to do so. For instance, the judgment

$$u:B; x_1:A_1, \dots, x_n:A_n \vdash P :: z:C \quad (1)$$

specifies a process P which offers behavior C along name z by making use of an *unrestricted* behavior B (a replicated server, available on name u) and of *linear* behaviors A_1, \dots, A_n (offered on names x_1, \dots, x_n). A process implementing one of these linear dependencies could be specified by the judgment

$$\cdot; \cdot \vdash S_1 :: x_1:A_1 \quad (2)$$

which says that process S_1 does not depend on any linear or unrestricted session behaviors to offer behavior A_1 along name x_1 . We write ‘ \cdot ’ to denote the empty set of dependencies. Given a typed interface such as (1), to satisfy each of the declared behavioral dependencies we need to first (i) compose the given process with another one which realizes the required behavior, and then (ii) restrict the name in which the behavior is required/offered, to avoid interferences. As a result, the interactions between the given process and the processes implementing its dependencies are unobservable. In the case of (1) and (2) above we would obtain the following typed composition:

$$u:B; x_2:A_2, \dots, x_n:A_n \vdash (\nu x_1)(S_1 \mid P) :: z:C \quad (3)$$

Hence, interactions on name x_1 become unobservable in the resulting composed process; its set of dependencies combines those of P (excepting $x_1:A_1$) and those of S_1 (in this case, the empty set). From (3) we could proceed similarly for all the behaviors declared in the left-hand side, thus obtaining a typed process without dependencies:

$$\cdot; \cdot \vdash (\nu \tilde{m})(!u(y).R \mid S_1 \mid \dots \mid S_n \mid P) :: z:C \quad (4)$$

with $\tilde{m} = u, x_1, \dots, x_n$ and $\cdot; \cdot \vdash R :: y:B$. In the above process, all behavioral dependencies arise as internal reductions; the only visible behavior takes place on name z . Notice that processes R, S_1, \dots, S_n may well have internal behavior on their own. For processes such as the one in (4), the interplay of confluence with strong normalization would be significant, as it could crucially ensure that session behavior as declared by judgments in the right-hand side ($z:C$ in this case) will be always offered, independently from any arbitrary interleaving of internal reductions from different sources.

Now, in sharp contrast to the normalizing, confluent nature of computation in many typed functional calculi, process calculi are inherently non-terminating, non-confluent models of concurrent computation. Hence, unsurprisingly, ensuring strong normalization and confluence in calculi for concurrency is a hard problem: in (variants of) the π -calculus, proofs require heavy constraints on the language and/or its types, often relying on ad-hoc machineries (see [15] for a survey on termination in process calculi). As a first challenge, we wonder: building upon our linear type structure, directly obtained from the Curry–Howard correspondence in [11], can we establish useful forms of strong normalization and confluence for session-typed, communicating processes?

While from an operational standpoint strong normalization and confluence are relevant, at a more foundational level they are also related to notions of *typed equality*. For instance, in the simply-typed λ -calculus, strong normalization and confluence ensure that normal forms exist and are unique, and entail decidability of denotational equality. In our concurrent setting, strong normalization is also related to *behavioral equivalence*—arguably the most basic notion in a theory of processes. Behavioral equivalences enable us to formally assert when two process terms denote the “same behavior”. A first, basic connection between strong normalization, confluence, and behavioral equivalence is obtained by means of subject reduction/type preservation: process behavior (as declared by typing judgments) is preserved along arbitrary reduction steps. Building upon this connection, any notion of behavioral equality over session-typed processes should be necessarily informed by the correspondence between session types and linear logic propositions. As detailed in [11], such a correspondence is realized by relating *proof conversions* in linear logic with appropriate notions in the process setting. Interestingly, by virtue of such proof conversions the correspondence already induces a notion of typed process equality. As illustration, consider the following process equalities, two instances of proof conversions:

$$(\nu x)(P \mid \bar{z}(y).(Q \mid R)) \simeq_c \bar{z}(y).((\nu x)(P \mid Q) \mid R) \quad (5)$$

$$x(y).z(w).P \simeq_c z(w).x(y).P \quad (\text{with } x \neq z) \quad (6)$$

In our framework, equality (5) results from the interplay of typed constructs for (bound) output (on name z) and process composition, whereas (6) arises from the typing of two independent sessions (on names x and z). Crucially, in both cases, the equated processes are syntactically very different and yet they are associated to the *same* typing judgment—that is, their typed session interface decrees the same behavior. As a second challenge, we ask: can we define a notion of typed process equality that is both natural and intuitive, that enjoys good properties (e.g., congruence), and that captures the notion of equality that is already induced by the logic interpretation via proof conversions?

A clear understanding of the status of strong normalization, confluence, and process equalities would provide a fundamental stepping stone towards a deeper understanding on how session types delineate communications. That is, basic behavioral equivalences over *equally typed* processes (in which strong normalization and confluence are expected to play substantial roles) may also provide a basis for reasoning about the behavior of processes with *different types*. In fact, given that session types represent service interfaces of distributed software artifacts, it is legitimate to ask whether the logic interpretation enables reasoning techniques at the level of session types. Such techniques appear very useful from a pragmatic perspective—for instance, they could enable natural notions of interface compatibility. Reasoning techniques at the level of types would also be useful from the more foundational standpoint of typed equality. To illustrate this, let us consider the session-typed interpretation of \otimes given in [11], whereby an object of type $A \otimes B$ denotes a session that first *outputs* a channel of type A and then behaves as B . This intuitive description may suggest an asymmetric interpretation, as opposed to the well-known symmetric nature of \otimes . This apparent asymmetry is already clarified in [11]: using a suitable typed process, it is shown how a session of type $A \otimes B$ may be *coerced* into one of type $B \otimes A$ (and viceversa). This justification, however, leaves open the general issue of equality over session types. In fact, we wish to understand the formal meaning in our setting of a notion of typed equality, in such a way that expected logic principles such as

$$A \otimes B \simeq B \otimes A \quad (7)$$

are properly justified. A final challenge would be then: building upon typed process equivalences, can we derive a simple notion of equality over session types that justifies/validates principles such as (7) above but also arbitrary interface transformations?

With the aim of addressing the challenges described above, the present paper offers the following technical contributions:

- (1) We present a simple theory of *logical relations* for session-typed processes, and use it to show that well-typed processes are both *strongly normalizing* and *confluent*.

The method of *logical relations* [43,44] has proved to be extremely productive in the functional setting; properties such as strong normalization and parametricity can be established via logical relations. Although the logic interpretation in [11] assigns types to names (and not to terms, as in the typed λ -calculus), quite remarkably, our *linear* logical relations are truly defined on the structure of types—as in logical relations for the typed λ -calculus [43,44]. This allows for simple proofs of strong normalization and confluence, which follow closely the principles of the (linear) type system. To our knowledge, ours are the first proofs of their kind in the context of session-based concurrency.

- (2) We investigate a *behavioral theory* for session-typed processes, defined as a typed contextual equivalence which follows the principles of the logical interpretation.

Well-studied in the untyped case, behavioral equivalences have been only little studied for session-typed processes (in fact, the only previous work we are aware of in the binary setting is [27]). We introduce *typed context bisimilarity*, a natural notion of observational equivalence for typed processes. We show how, thanks to the combination of type preservation, progress, strong normalization, and confluence, typed context bisimilarity satisfies τ -*inertness*, as studied by Groote and Sellink [20]. Intuitively, τ -inertness says that reduction (internal behavior) does not change process behavior. This is most relevant for verification, as it means that our well-typed processes can perform arbitrarily many

reductions while remaining in the same equivalence class. In our setting, this guarantee is neatly complemented by strong normalization, which ensures finitely many reductions.

(3) By relying on the above results, we then develop two *applications*, which clarify further the nature of the logical interpretation of session types put forward in [11]:

- We prove that proof conversions are *sound* with respect to observational equivalence. This way, processes equalities induced by proof conversions (such as (5) and (6) above) correspond to typed context bisimilarities. This soundness result elegantly explains subtle forms of causality that arise in the execution of concurrent sessions.
- Building upon typed bisimilarity, we offer a characterization of *type isomorphisms* (see, e.g., [18]). Intuitively, such isomorphisms result from linear logic equivalences which are realized by process coercions. Our characterization allows us to show that principles such as (7) above are indeed isomorphisms.

Our applications thus shed further light on the relationship between linear logic and structured communications. Strong normalization and confluence properties are central in the associated coinductive reasoning, intuitively because in the bisimulation game strong transitions are always matched by weak transitions with finite and confluent internal behavior.

Organization Next, in Section 2, we present our process model, a synchronous π -calculus with guarded choice. Section 3 recalls the type system derived from the logical interpretation and main results from [11]. Section 4 presents proof conversions, describing inference permutability issues derived from the logical interpretation. Section 5 presents linear logical relations for typed processes, as well as the proof of strong normalization and confluence. Section 6 introduces typed context bisimilarity and studies its main properties. Section 7 presents our two applications. Finally, Section 8 discusses related work, and Section 9 collects some final remarks. A number of proofs and technical details have been moved to the Appendix.

This paper is an extended version of the conference paper [34]. In this presentation, we provide full technical details and include some new material: Section 4, on proof conversions; Section 5.3, on a proof of confluence via linear logical relations; and the proof of τ -inertness given in Section 6.

2. Process model: syntax and semantics

We introduce the syntax and operational semantics of the synchronous π -calculus [42] extended with (binary) guarded choice.

Definition 2.1 (*Processes*). Given an infinite set Λ of *names* (ranged over x, y, z, u, v), the set of *processes* (ranged over P, Q, R) is defined by

$$P ::= \mathbf{0} \mid P \mid Q \mid (\nu y)P \mid \bar{x}y.P \mid x(y).P \mid !x(y).P \\ \mid [x \leftrightarrow y] \mid x.\text{inl}; P \mid x.\text{inr}; P \mid x.\text{case}(P, Q).$$

The operators $\mathbf{0}$, $P \mid Q$, and $(\nu y)P$ comprise the static fragment of any π -calculus: they represent inaction, parallel composition, and name restriction, respectively. We then have prefixed processes $\bar{x}y.P$ and $x(y).P$: while the former denotes a process which sends name y on x and then proceeds as P , the latter denotes a process which receives a name z on x , and then proceeds as P with parameter y replaced by z . Process $!x(y).P$ denotes replicated (persistent) input. Following [40], we write $\bar{x}(y)$ as an abbreviation for $(\nu y)\bar{x}y$. The *forwarding construct* $[x \leftrightarrow y]$ equates names x and y ; it is a primitive representation of a copycat process, akin to the link processes used in internal mobility encodings of name-passing [5]. As described in Section 3, this construct allows for a simple identity axiom in the type system [45]. The remaining three operators define a minimal labeled choice mechanism, comparable to the n -ary branching constructs found in standard session π -calculi (see, e.g., [23]). Without loss of generality we restrict our model to binary choice. In restriction $(\nu y)P$ and input $x(y).P$ the distinguished occurrence of name y is binding, with scope P .

The set of *free names* of a process P is denoted by $fn(P)$. A process is *closed* if it does not contain free occurrences of names. We identify process up to consistent renaming of bound names, writing \equiv_α for this congruence. We write $P\{x/y\}$ for the capture-avoiding substitution of x for y in P . While *structural congruence* expresses basic identities on the structure of processes, *reduction* expresses the behavior of processes.

Definition 2.2. *Structural congruence* ($P \equiv Q$) is the least congruence relation on processes such that

$$\begin{array}{ll} P \mid \mathbf{0} \equiv P & P \equiv_\alpha Q \Rightarrow P \equiv Q \\ P \mid Q \equiv Q \mid P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\ (\nu x)\mathbf{0} \equiv \mathbf{0} & x \notin fn(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) \\ (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & [x \leftrightarrow y] \equiv [y \leftrightarrow x]. \end{array}$$

Definition 2.3. *Reduction* ($P \rightarrow Q$) is the binary relation on processes defined by the rules in Fig. 1.

By definition, reduction is closed under \equiv . It specifies the computations a process performs on its own. To define the interactions of a process with its environment, we extend the early transition system for the π -calculus [42] with labels

$x(y).Q \mid x(z).P \rightarrow Q \mid P\{y/z\}$	$x.inr; P \mid x.case(Q, R) \rightarrow P \mid R$
$x(y).Q \mid !x(z).P \rightarrow Q \mid P\{y/z\} \mid !x(z).P$	$x.inl; P \mid x.case(Q, R) \rightarrow P \mid Q$
$(\nu x)([x \leftrightarrow y] \mid P) \rightarrow P\{y/x\} \quad (y \notin fn(P))$	$P \rightarrow Q \Rightarrow (\nu y)P \rightarrow (\nu y)Q$
$P \equiv P', P' \rightarrow Q', Q' \equiv Q \Rightarrow P \rightarrow Q$	$Q \rightarrow Q' \Rightarrow P \mid Q \rightarrow P \mid Q'$

Fig. 1. Process reduction.

(out)	(in)	(id)
$\bar{x}y.P \xrightarrow{\bar{x}y} P$	$x(y).P \xrightarrow{x(z)} P\{z/y\}$	$(\nu x)([x \leftrightarrow y] \mid P) \xrightarrow{\tau} P\{y/x\}$
(par)	(com)	(res)
$\frac{P \xrightarrow{\alpha} Q \quad bn(\alpha) \cap fn(R) = \emptyset}{P \mid R \xrightarrow{\alpha} Q \mid R}$	$\frac{P \xrightarrow{\bar{\alpha}} P' \quad Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	$\frac{P \xrightarrow{\alpha} Q \quad y \notin fn(\alpha)}{(\nu y)P \xrightarrow{\alpha} (\nu y)Q}$
(open)	(close)	
$\frac{P \xrightarrow{\bar{x}y} Q}{(\nu y)P \xrightarrow{\bar{x}(y)} Q}$	$\frac{P \xrightarrow{\bar{x}(y)} P' \quad Q \xrightarrow{x(y)} Q' \quad y \notin fn(Q)}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')}$	
(rep)	(lout)	(rout)
$!x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P$	$x.inl; P \xrightarrow{\bar{x}.inl} P$	$x.inr; P \xrightarrow{\bar{x}.inr} P$
(lin)	(rin)	
$x.case(P, Q) \xrightarrow{x.inl} P$	$x.case(P, Q) \xrightarrow{x.inr} Q$	

Fig. 2. π -Calculus labeled transition system.

and transition rules for the choice and forwarding constructs. A transition $P \xrightarrow{\alpha} Q$ denotes that P may evolve to Q by performing the action represented by label α . Labels are given by:

$$\alpha ::= x(y) \mid \bar{x}y \mid \bar{x}(y) \mid x.inl \mid \bar{x}.inl \mid x.inr \mid \bar{x}.inr \mid \tau$$

Actions are input $x(y)$, the left/right offers $x.inl$ and $x.inr$, and their matching co-actions, respectively, the free output $\bar{x}y$ and bound output $\bar{x}(y)$ actions, and the left/ right selections $\bar{x}.inl$ and $\bar{x}.inr$. The bound output $\bar{x}(y)$ denotes extrusion of a fresh name y along x . Internal action is denoted by τ . In general, an action α (resp. $\bar{\alpha}$) requires a matching $\bar{\alpha}$ (resp. α) in the environment to enable progress, as specified by the transition rules. For a label α , we define the sets $fn(\alpha)$ and $bn(\alpha)$ of free and bound names, respectively, as usual. We denote by $s(\alpha)$ the subject of α (e.g., x in $\bar{x}(y)$).

Definition 2.4 (Labeled transition system). The relation labeled transition ($P \xrightarrow{\alpha} Q$) is defined by the rules in Fig. 2, together with the symmetric versions of rules (par), (com), and (close).

Hence, rules in Fig. 2 extend standard rules for the π -calculus with rules for the forwarding construct (cf. rule (id)) and the labeled choice operator (cf. rules (lout), (rout), (lin), and (rin)). Weak transitions are also defined as usual. Let us write \Rightarrow for the reflexive, transitive closure of $\xrightarrow{\tau}$ and $\rho_1 \rho_2$ for the composition of relations ρ_1, ρ_2 . Given $\alpha \neq \tau$, notation $\xRightarrow{\alpha}$ stands for $\Rightarrow \xrightarrow{\alpha} \Rightarrow$ and $\xRightarrow{\tau}$ stands for \Rightarrow .

We close this section by recalling some basic facts about reduction, structural congruence, and labeled transition, namely closure of labeled transitions under structural congruence, and coincidence of τ -labeled transition and reduction:

Proposition 2.1. (See [42].) Let P be a π -calculus process.

- (1) If $P \equiv^{\alpha} Q$ then $P \xrightarrow{\alpha} Q$.
- (2) $P \rightarrow Q$ if and only if $P \xRightarrow{\tau} Q$.

3. Session types as intuitionistic linear logic propositions

As anticipated in the introduction, the type structure coincides with intuitionistic linear logic [19,2], omitting atomic formulas and the additive constants \top and $\mathbf{0}$.

Definition 3.1 (Types). Types (A, B, C) are given by

$$A, B ::= \mathbf{1} \mid !A \mid A \otimes B \mid A \multimap B \mid A \& B \mid A \oplus B.$$

Types are assigned to (session) channels/names, and are interpreted as a form of session types; an assignment $x:A$ enforces the use of name x according to discipline A . We use $A \otimes B$ to type a channel that first performs an output to its partner (sending a session channel of type A) before proceeding as specified by B . Similarly, $A \multimap B$ types a channel that

first performs an input from its partner (receiving a session channel of type A) before proceeding as specified by B . Type $\mathbf{1}$ represents a terminated session, no further interaction will take place on it; names of type $\mathbf{1}$ may still be passed around in sessions, as opaque values. $A \& B$ types a channel that offers its partner a choice between an A behavior (“left” choice) and a B behavior (“right” choice). Dually, $A \oplus B$ types a session that either selects “left” and then proceeds as specified by A , or else selects “right”, and then proceeds as specified by B . Type $!A$ types a shared (non-linear) channel, to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type A .

A *type environment* is a set of type assignments of the form $x:A$, where A is a type and x is a name, the names being pairwise disjoint. Type environments may be subject to different structural principles. There are at least three sensible principles. The first one, *exchange*, indicates that the ordering of type assignments does not matter. The second principle, *weakening*, says that type assignments need not be used. Finally, the *contraction* principle says that type assignments may be duplicated. We write Δ for the *linear* type environment, subject only to exchange; we write Γ for the *unrestricted* type environment, subject to exchange, weakening, and contraction principles.

A type judgment is of the form

$$\Gamma; \Delta \vdash P :: z:C \quad (8)$$

where name declarations in Γ are always propagated unchanged to all premises in the typing rules, while name declarations in Δ are handled multiplicatively or additively, depending on the nature of the connective being defined. The domains of Γ, Δ and $z:C$ are required to be pairwise disjoint. The judgment in (8) asserts: P is ensured to safely provide a usage of name z according to the behavior specified by type C , whenever composed with any process environment providing usages of names according to the behaviors specified by assignments in Γ and Δ .

Our typing judgment defines an intuitive reading of processes. Given (8), process P represents a system providing behavior C at channel z , building on “services” declared in Γ and Δ . This way, for instance, a client Q that relies on external services and does not provide any would be typed as $\Gamma; \Delta \vdash Q :: -:\mathbf{1}$, where we write $-$ to denote a “dummy name” that does not occur in Q . A system typed as $\Gamma; \Delta \vdash R :: z:!A$ represents a shared server. Interestingly, the asymmetry induced by the intuitionistic interpretation of $!$ enforces locality of shared names but not of linear (session names), which exactly corresponds to the intended model of sessions.

Notation 3.1. When empty, environments Γ and Δ are denoted by ‘ \cdot ’. Also, we often use T for right-hand side singleton environments (e.g., $z:C$). Furthermore, we sometimes write $\vdash P :: T$ to stand for $\cdot; \cdot \vdash P :: T$.

We briefly comment on the rules that define the typing judgment, given in Fig. 3. Rule (Tid) defines identity in terms of the forwarding construct. The so-called left rules (marked with L) define how to use a session of a given type, whereas the right rules (marked with R) define how to offer a session of a given type. The type $\mathbf{1}$ which is associated with the lack of observable behavior is offered by the inactive process $\mathbf{0}$, as specified in rule (T1R). Using such a session thus requires no behavior (rule T1L).

Offering a session of type $x:A \otimes B$, as specified in rule (T \otimes R), is achieved by performing the output of a fresh name y along x , which will offer the session behavior A specified by P , after which Q will then offer the session behavior $x:B$. Since in rule (T \otimes R) the sent name is always fresh, our typed calculus conforms to an internal mobility discipline [5,40], without loss of expressiveness. Note how P and Q must use disjoint sets of ambient sessions. Using a session of such a type is achieved by the corresponding input behavior. The rules for \multimap are dual to those of \otimes : offering $x:A \multimap B$ is achieved by performing an input of y along x , after which y is used according to the session behavior A and x offers the session behavior B . Using such a session is achieved by providing the appropriate output actions.

The cut rules (Tcut) and (Tcut^l) define typed linear and unrestricted session composition, respectively. They follow the “composition plus hiding” principle [1], extended to a name-passing setting. Other linear typing rules for parallel composition are derivable—see [11]. Relevant to our current development is the following derivable rule, defining *independent parallel composition*:

$$\text{(INDCOMP)} \quad \frac{\Gamma; \Delta_1 \vdash P :: -:\mathbf{1} \quad \Gamma; \Delta_2 \vdash Q :: T}{\Gamma; \Delta_1, \Delta_2 \vdash P \mid Q :: T}$$

Rule (T!R) specifies that offering a session of type $!A$ requires a replicated input along x , where each replica will offer the behavior A along the received name (without using any linear sessions). Offering an external choice, as specified by rule (T&R), is achieved by a *case* prefix, which waits for the choice, proceeding accordingly. Using such a session is achieved by performing the appropriate selections (rules (T&L₁) and (T&L₂)). The rules for internal choice \oplus are dual.

We always consider processes modulo structural congruence; hence, typability is closed under \equiv by definition. πDILL enjoys the usual properties of equivariance, weakening, and contraction in Γ . The coverage property also holds: if $\Gamma; \Delta \vdash P :: z:A$ then $\text{fn}(P) \subseteq \Gamma \cup \Delta \cup \{z\}$. In the presence of type-annotated restrictions $(\nu x:A)P$, as usual in typed π -calculi [42], type-checking is decidable.

Session type constructors thus correspond directly to intuitionistic linear logic connectives. By erasing processes, typing judgments in πDILL correspond to DILL , a sequent formulation of Barber’s dual intuitionistic linear logic [2,13]. Below we informally recall this correspondence; see [11,12] for details.

(Tid)	(T1L)	(T1R)
$\frac{}{\Gamma; x:A \vdash [x \leftrightarrow z] :: z:A}$	$\frac{\Gamma; \Delta \vdash P :: T}{\Gamma; \Delta, x:1 \vdash P :: T}$	$\frac{}{\Gamma; \cdot \vdash 0 :: x:1}$
(T \otimes L)	(T \otimes R)	
$\frac{\Gamma; \Delta, y:A, x:B \vdash P :: T}{\Gamma; \Delta, x:A \otimes B \vdash x(y).P :: T}$	$\frac{\Gamma; \Delta \vdash P :: y:A \quad \Gamma; \Delta' \vdash Q :: x:B}{\Gamma; \Delta, \Delta' \vdash \bar{x}(y).(P \mid Q) :: x:A \otimes B}$	
(T \multimap L)	(T \multimap R)	
$\frac{\Gamma; \Delta \vdash P :: y:A \quad \Gamma; \Delta', x:B \vdash Q :: T}{\Gamma; \Delta, \Delta', x:A \multimap B \vdash \bar{x}(y).(P \mid Q) :: T}$	$\frac{\Gamma; \Delta, y:A \vdash P :: x:B}{\Gamma; \Delta \vdash x(y).P :: x:A \multimap B}$	
(Tcut)	(Tcut')	
$\frac{\Gamma; \Delta \vdash P :: x:A \quad \Gamma; \Delta', x:A \vdash Q :: T}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: T}$	$\frac{\Gamma; \cdot \vdash P :: y:A \quad \Gamma, u:A; \Delta \vdash Q :: T}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: T}$	
(T!L)	(Tcopy)	(T!R)
$\frac{\Gamma, u:A; \Delta \vdash P\{u/x\} :: T}{\Gamma; \Delta, x:!A \vdash P :: T}$	$\frac{\Gamma, u:A; \Delta, y:A \vdash P :: T}{\Gamma, u:A; \Delta \vdash \bar{u}(y).P :: T}$	$\frac{\Gamma; \cdot \vdash Q :: y:A}{\Gamma; \cdot \vdash !x(y).Q :: x:!A}$
(T \oplus L)	(T \oplus R)	
$\frac{\Gamma; \Delta, x:A \vdash P :: T \quad \Gamma; \Delta, x:B \vdash Q :: T}{\Gamma; \Delta, x:A \oplus B \vdash x.\text{case}(P, Q) :: T}$	$\frac{\Gamma; \Delta \vdash P :: x:A \quad \Gamma; \Delta \vdash Q :: x:B}{\Gamma; \Delta \vdash x.\text{case}(P, Q) :: x:A \oplus B}$	
(T&L ₁)	(T \oplus R ₁)	
$\frac{\Gamma; \Delta, x:A \vdash P :: T}{\Gamma; \Delta, x:A \& B \vdash x.\text{inl}; P :: T}$	$\frac{\Gamma; \Delta \vdash P :: x:A}{\Gamma; \Delta \vdash x.\text{inl}; P :: x:A \oplus B}$	
(T&L ₂)	(T \oplus R ₂)	
$\frac{\Gamma; \Delta, x:B \vdash P :: T}{\Gamma; \Delta, x:A \& B \vdash x.\text{inr}; P :: T}$	$\frac{\Gamma; \Delta \vdash P :: x:B}{\Gamma; \Delta \vdash x.\text{inr}; P :: x:A \oplus B}$	

Fig. 3. The type system πDILL .

DILL is equipped with a faithful proof term assignment: sequents have the form

$$\Gamma; \Delta \vdash D : C \quad (9)$$

where Γ is the unrestricted context, Δ the linear context, C a formula (i.e., a type), and D the proof term that faithfully represents the derivation of $\Gamma; \Delta \vdash C$. Given the parallel structure of the two systems, if $\Gamma; \Delta \vdash D : A$ is derivable in DILL then there is a process P and a name z such that $\Gamma; \Delta \vdash P :: z:A$ is derivable in πDILL . The converse also holds: if $\Gamma; \Delta \vdash P :: z:A$ is derivable in πDILL there is a derivation D that proves $\Gamma; \Delta \vdash D : A$. This correspondence is made explicit by a translation from faithful proof terms to processes: given $\Gamma; \Delta \vdash D : C$, we write \hat{D}^z for the translation of D such that $\Gamma; \Delta \vdash \hat{D}^z :: z:C$.

More precisely, we have *typed extraction*: we write

$$\Gamma; \Delta \vdash D \rightsquigarrow P :: z:A \quad (10)$$

meaning “proof D extracts to P ”, whenever $\Gamma; \Delta \vdash D : A$ and $\Gamma; \Delta \vdash P :: z:A$ and $P \equiv \hat{D}^z$. Typed extraction is unique up to structural congruence. As processes are related by structural and computational rules, namely those involved in the definition of \equiv and \rightarrow , derivations in DILL are related by structural and computational rules, that express certain sound proof transformations that arise in cut elimination. Reductions generally take place when a right rule meets a left rule for the same connective, and correspond to reduction steps in the process term assignment. Similarly, structural conversions in DILL correspond to structural equivalences in the π -calculus, since they just change the order of cuts.

We now recall some results from [11,12], on *subject reduction* (type preservation) and *progress* (deadlock-freedom) for well-typed processes. For any P , define *live*(P) iff $P \equiv (\nu \tilde{n})(\pi.Q \mid R)$, for some sequence of names \tilde{n} , a process R , and a *non-replicated* guarded process $\pi.Q$.

Theorem 3.1 (Subject reduction). *If $\Gamma; \Delta \vdash P :: z:A$ and $P \rightarrow Q$ then $\Gamma; \Delta \vdash Q :: z:A$.*

Lemma 3.1. *Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:C$. If *live*(P) then there is a Q such that either (1) $P \rightarrow Q$, or (2) $P \xrightarrow{\alpha} Q$ for α where $s(\alpha) \in (z, \Gamma, \Delta)$. Moreover, if $C = !A$ for some A , then $s(\alpha) \neq z$.*

Theorem 3.2 (Progress). *If $\cdot; \cdot \vdash P :: z:1$ and *live*(P) then exists a Q such that $P \rightarrow Q$.*

We close this section recalling some other auxiliar results from [11,12].

$$\begin{aligned}
(\mathbf{v}x)(\hat{D} \mid \bar{z}(y).(\hat{E} \mid \hat{F})) &\simeq_c \bar{z}(y).((\mathbf{v}x)(\hat{D} \mid \hat{E}) \mid \hat{F}) & (11) \\
(\mathbf{v}x)(\hat{D} \mid y(z).\hat{E}) &\simeq_c y(z).(\mathbf{v}x)(\hat{D} \mid \hat{E}) & (12) \\
(\mathbf{v}x)(\hat{D} \mid y.\text{inl}; \hat{E}) &\simeq_c y.\text{inl}; (\mathbf{v}x)(\hat{D} \mid \hat{E}) & (13) \\
(\mathbf{v}x)(\hat{D} \mid \bar{u}(y).\hat{E}) &\simeq_c \bar{u}(y).(\mathbf{v}x)(\hat{D} \mid \hat{E}) & (14) \\
(\mathbf{v}x)(\hat{D} \mid y.\text{case}(\hat{E}, \hat{F})) &\simeq_c y.\text{case}((\mathbf{v}x)(\hat{D} \mid \hat{E}), (\mathbf{v}x)(\hat{D} \mid \hat{F})) & (15) \\
(\mathbf{v}u)(!u(y).\hat{D} \mid \mathbf{0}) &\simeq_c \mathbf{0} & (16) \\
(\mathbf{v}u)(!u(y).\hat{D} \mid \bar{x}(z).(\hat{E} \mid \hat{F})) &\simeq_c \bar{x}(z).((\mathbf{v}u)(!u(y).\hat{D} \mid \hat{E}) \mid (\mathbf{v}u)(!u(y).\hat{D} \mid \hat{F})) & (17) \\
(\mathbf{v}u)((!u(y).\hat{D}) \mid y(z).\hat{E}) &\simeq_c y(z).(\mathbf{v}u)((!u(y).\hat{D}) \mid \hat{E}) & (18) \\
(\mathbf{v}u)((!u(z).\hat{D}) \mid y.\text{inl}; \hat{E}) &\simeq_c y.\text{inl}; (\mathbf{v}u)((!u(z).\hat{D}) \mid \hat{E}) & (19) \\
(\mathbf{v}u)(!u(z).\hat{D} \mid y.\text{case}(\hat{E}, \hat{F})) &\simeq_c y.\text{case}((\mathbf{v}u)(!u(z).\hat{D} \mid \hat{E}), (\mathbf{v}u)(!u(z).\hat{D} \mid \hat{F})) & (20) \\
(\mathbf{v}u)(!u(y).\hat{D} \mid !x(z).\hat{E}) &\simeq_c !x(z).(\mathbf{v}u)(!u(y).\hat{D} \mid \hat{E}) & (21) \\
(\mathbf{v}u)(!u(y).\hat{D} \mid \bar{v}(y).\hat{E}) &\simeq_c \bar{v}(y).(\mathbf{v}u)(!u(y).\hat{D} \mid \hat{E}) & (22) \\
\bar{z}(w).(\hat{F} \mid \bar{x}(y).(\hat{D} \mid \hat{E})) &\simeq_c \bar{x}(y).(\hat{D} \mid \bar{z}(w).(\hat{F} \mid \hat{E})) & (23) \\
x(y).z(w).\hat{D} &\simeq_c z(w).x(y).\hat{D} & (24)
\end{aligned}$$

Fig. 4. A sample of process equalities induced by proof conversions (cf. Definition 4.1).

Lemma 3.2 (Action characterization lemmas, excerpt). Let $\Gamma; \Delta \vdash D \rightsquigarrow P :: x:C$. Then we have:

1. If $P \xrightarrow{\alpha} Q$ and $C = \mathbf{1}$ then $s(\alpha) \neq x$.
2. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $C = A \otimes B$ then $\alpha = \bar{x}(y)$.
3. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $C = A \multimap B$ then $\alpha = x(y)$.
4. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $C = A \& B$ then $\alpha = x.\text{inl}$ or $\alpha = x.\text{inr}$.
5. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $C = A \oplus B$ then $\alpha = \bar{x}.\text{inl}$ or $\alpha = \bar{x}.\text{inr}$.
6. If $P \xrightarrow{\alpha} Q$ and $s(\alpha) = x$ and $C = !A$ then $\alpha = x(y)$.

Lemma 3.3 (Preservation lemma, output case). Assume

- $\Gamma; \Delta_1 \vdash D \rightsquigarrow P :: x:A_1 \otimes A_2$ with $P \xrightarrow{\bar{x}(y)} P'$; and
- $\Gamma; \Delta_2, x:A_1 \otimes A_2 \vdash E \rightsquigarrow Q :: z:C$ with $Q \xrightarrow{x(y)} Q'$.

Then: $\Gamma; \Delta_1, \Delta_2 \vdash F \rightsquigarrow R :: z:C$ for $R \equiv (\mathbf{v}y)(\mathbf{v}x)(P' \mid Q')$.

Lemma 3.4. Assume $\Gamma; \Delta \vdash D \rightsquigarrow P :: z:C$ and not $\text{live}(P)$. Then

1. $C = \mathbf{1}$ or $C = !C'$, for some C' ;
2. $(x_i : A_i) \in \Delta$ implies $A_i = \mathbf{1}$ or there is B_j with $A_i = !B_j$;
3. $C = !C'$ implies $P \equiv (\mathbf{v}\tilde{x})(!z(y).R \mid R')$

4. Inference permutability and proof conversions

Derivations in DILL are related by structural and computational rules that express sound proof transformations that arise in cut elimination. As mentioned in Section 3 (and detailed in [11]), in our interpretation reductions and structural conversions in DILL correspond to reductions and structural congruence in the π -calculus. There is, however, a group of conversions in DILL not considered in [11] and which do not correspond to neither reduction or structural congruence in the process side. We call them *proof conversions*: they induce a congruence on typed processes, denoted by \simeq_c .

This section illustrates proof conversions and their associated π -calculus processes. Fig. 4 presents a sample of process equalities extracted from them; the full list is reported in Appendix D. Each equality $P \simeq_c Q$ in the figure is associated to appropriate right- and left-hand side typings; this way, e.g., equality (24) in Fig. 4—related to two applications of rule (T \otimes L)—could be stated as

$$\Gamma; \Delta, x:A \otimes B, z:C \otimes D \vdash x(y).z(w).P \simeq_c z(w).x(y).P :: T$$

where Γ and Δ are environments, A, B, C, D are types, and T is a right-hand side typing. For the sake of illustration, however, in Fig. 4 these typings are elided, as we would like to stress on the consequences of conversions on the process side. Proof conversions describe the interplay of two rules in a type-preserving way: regardless of the order in which the two rules are applied, they lead to typing derivations with the same right- and left-hand side typings, but with syntactically different processes.

We first formally introduce the set of process equalities induced by proof conversions. Then, we provide intuitions about how such conversions arise in our setting.

Definition 4.1 (*Proof conversions*). We define \simeq_c as the least congruence on processes induced by the process equalities in Figs. D.6, D.7, D.8, and D.9 (Pages 300–301).

We classify proof conversions into five classes, denoted by (A)–(E):

- (A) *Permutations between rule (Tcut) and a right or left rule*. This class of conversions represents the interaction of a process offering a service C on x , with some process requiring such service; this process varies according to the particular rule considered. As an example, the following inference represents the interplay of rules (T \multimap L) and (Tcut):

$$\frac{\frac{\Gamma; \Delta_2, x:C \vdash Q :: z:A \quad \Gamma; \Delta_3, y:B \vdash R :: T}{\Gamma; \Delta_2, \Delta_3, x:C, y:A \multimap B \vdash \bar{y}(z).(Q \mid R) :: T} \text{ (T}\multimap\text{L)}}{\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(P \mid \bar{y}(z).(Q \mid R)) :: T} \text{ (Tcut)}$$

where $\Delta = \Delta_1, \Delta_2, \Delta_3$. Permutability is justified by the following inference:

$$\frac{\frac{\Gamma; \Delta_1 \vdash P :: x:C \quad \Gamma; \Delta_2, x:C \vdash Q :: z:A}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P \mid Q) :: z:A} \text{ (Tcut)}}{\Gamma; \Delta, y:A \multimap B \vdash \bar{y}(z).((\nu x)(P \mid Q) \mid R) :: T} \text{ (T}\multimap\text{L)}$$

This class of permutations is given by process equalities in Fig. D.6.

- (B) *Permutations between rule (Tcut) and a left rule*. In contrast to permutations in class (A), this class of conversions represents the interaction of a process requiring a service C on x , with some process offering such a service. This distinction is due to the shape of rule (Tcut). To see the difference with the permutations in class (A), consider the inferences given above with those for the permutation below, which also concerns rules (T \multimap L) and (Tcut). Letting $\Delta = \Delta_1, \Delta_2, \Delta_3$, we have:

$$\frac{\frac{\Gamma; \Delta_1 \vdash P :: z:A \quad \Gamma; \Delta_2, y:B \vdash Q :: x:C}{\Gamma; \Delta_1, \Delta_2, y:A \multimap B \vdash \bar{y}(z).(P \mid Q) :: x:C} \text{ (T}\multimap\text{L)}}{\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(\bar{y}(z).(P \mid Q) \mid R) :: T} \text{ (Tcut)}$$

Permutability is then justified by the following inference:

$$\frac{\frac{\Gamma; \Delta_2, y:B \vdash Q :: x:C \quad \Gamma; \Delta_3, x:C \vdash R :: T}{\Gamma; \Delta_2, \Delta_3, y:B \vdash (\nu x)(Q \mid R) :: T} \text{ (Tcut)}}{\Gamma; \Delta, y:A \multimap B \vdash \bar{y}(z).(P \mid (\nu x)(Q \mid R)) :: T} \text{ (T}\multimap\text{L)}$$

This class of permutations is given by process equalities in Fig. D.6.

- (C) *Permutations between rule (Tcut[!]) and a right or left rule*. This class of permutations is analogous to the two classes (A) and (B), but considering rule (Tcut[!]) instead of (Tcut). As an example, the following permutation concerns the interplay of rule (Tcut[!]) with rule (T \oplus R₁):

$$\frac{\frac{\Gamma; \cdot \vdash P :: y:C \quad \Gamma, u:C; \Delta \vdash Q :: z:A}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid Q) :: z:A} \text{ (Tcut}^!\text{)}}{\Gamma; \Delta \vdash z.\text{inl}; (\nu u)(!u(y).P \mid Q) :: z:A \oplus B} \text{ (T}\oplus\text{R}_1\text{)}$$

Then, permutability is justified by the following inference:

$$\frac{\Gamma; \cdot \vdash P :: y:C \quad \Gamma, u:C; \Delta \vdash Q :: z:A}{\Gamma; \Delta \vdash (\nu u)(!u(y).P \mid z.\text{inl}; Q) :: z:A \oplus B} \text{ (T}\oplus\text{R}_1\text{)} \quad \text{ (Tcut}^!\text{)}$$

This class of permutations is given by process equalities in Fig. D.7.

- (D) *Permutations between two left rules.* Classes (A)–(C) consider permutations in which one of the involved rules is some form of cut. Permutations which do not involve cuts are also possible; they represent type-preserving transformations for prefixes corresponding to independent (non-interfering) sessions. Here we consider permutations involving two left rules; that is, permutations in this class involve two different behavioral dependencies. As an example, the permutation below concerns the interplay of rule $(T\oplus L)$ with rule $(T\otimes L)$:

$$\frac{\frac{\Gamma; \Delta, z:C, x:B, y:A \vdash P :: T}{\Gamma; \Delta, z:C, x:A \otimes B \vdash x(y).P :: T} (T\otimes L) \quad \frac{\Gamma; \Delta, z:D, x:B, y:A \vdash Q :: T}{\Gamma; \Delta, z:D, x:A \otimes B \vdash x(y).Q :: T} (T\otimes L)}{\Gamma; \Delta, z:C \oplus D, x:A \otimes B \vdash z.\text{case}(x(y).P, x(y).Q) :: T} (T\oplus L)$$

Then, permutability is justified by the following inference:

$$\frac{\frac{\Gamma; \Delta, z:C, x:B, y:A \vdash P :: T \quad \Gamma; \Delta, z:D, x:B, y:A \vdash Q :: T}{\Gamma; \Delta, z:C \oplus D, x:B, y:A \vdash z.\text{case}(P, Q) :: T} (T\oplus L)}{\Gamma; \Delta, z:C \oplus D, x:A \otimes B \vdash x(y).z.\text{case}(P, Q) :: T} (T\oplus L)$$

This class is given by process equalities in Fig. D.8.

- (E) *Permutations between a left and a right rule.* This class of permutations also involves rules acting on two independent sessions: one rule acts on the left-hand side of the derivation (a behavioral dependence) while the other acts on the right-hand side (a behavioral offer). As an example, the permutation below concerns the interplay of rules $(T\&L_1)$ and $(T\otimes R)$:

$$\frac{\frac{\Gamma; \Delta_1, z:C \vdash P :: y:A \quad \Gamma; \Delta_2 \vdash Q :: x:B}{\Gamma; \Delta, z:C \vdash \bar{x}(y).(P \mid Q) :: x:A \otimes B} (T\otimes R)}{\Gamma; \Delta, z:C \& D \vdash z.\text{inl}; \bar{x}(y).(P \mid Q) :: x:A \otimes B} (T\&L_1)$$

where $\Delta = \Delta_1, \Delta_2$. Permutability is justified by the following inference:

$$\frac{\frac{\Gamma; \Delta_1, z:C \vdash P :: z:A}{\Gamma; \Delta, z:C \& D \vdash z.\text{inl}; P :: y:A} (T\&L_1) \quad \Gamma; \Delta_2 \vdash Q :: x:B}{\Gamma; \Delta, z:C \& D \vdash \bar{x}(y).(z.\text{inl}; P \mid Q) :: x:A \otimes B} (T\otimes R)$$

This class is given by process equalities in Fig. D.9.

Having recalled the type system and the permutability issues derived from the logical interpretation, the following sections investigate linear logical relations and observational equivalences for well-typed π -calculus processes.

5. Linear logical relations for session-typed processes

Here we introduce a theory of *linear* logical relations for session types, and use it to prove that well-typed processes are strongly normalizing (Definition 5.1) and confluent (Definition 5.2). The proof can be divided into major two steps:

- (1) Definition of a logical predicate on processes, by induction on the structure of (session) types. By definition, processes in the predicate are strongly normalizing (resp. confluent).
- (2) Prove that every well-typed process is in the logical predicate.

Concerning (1), we define the logical predicates by characterizing the general behavior of processes, as specified by their typing. To this end, both predicates (Definitions 5.4 and 5.7) are almost identical, with the only fundamental difference being the property of interest that we wish to capture. This highlights the generality of our proof technique.

In order to achieve (2) we must first show that the logical predicates satisfy certain fundamental closure conditions. Specifically, we must show that the predicates are closed under reduction (Propositions 5.10 and 5.19) and that a form of backward reduction closure (Propositions 5.11 and 5.20) also holds. The need for these intermediate results is common in logical relations developments, due to the fact that the logical predicates capture the extensional behavior of processes, as specified by their typings, satisfying the property of interest (i.e., strong normalization and confluence, respectively) and must therefore be invariant under unobservable actions. Given these basic soundness properties of the logical predicate it is then possible to show the main result that all well-typed processes satisfy the predicate.

5.1. Preliminaries

We begin by defining *strong normalization*. In some previous works in process calculi [41,15], strong normalization is simply referred to as *termination*. In what follows, we often use the two terms interchangeably. Below, we write $P \not\rightarrow$ to mean that P cannot reduce; it may perform visible actions, though.

Definition 5.1 (Termination). A process P *terminates*, noted $P \Downarrow$, if either $P \not\rightarrow$ or for any P' such that $P \rightarrow P'$ we have that $P' \Rightarrow P'' \not\rightarrow$.

We now define *confluence*. Our notion considers only weak transitions based on internal behavior; as such, it is closer to definitions of confluence used for functional languages than to the definitions used in process calculi, which consider labeled transitions. (We refer to Section 8 for additional comparisons.)

Definition 5.2 (Confluence). A process P is *confluent*, noted $P \Diamond$, if for any P_1, P_2 such that $P \Rightarrow P_1$ and $P \Rightarrow P_2$, there exists a P' such that $P_1 \Rightarrow P'$ and $P_2 \Rightarrow P'$.

We now state an extension to \equiv , which will be useful in our developments.

Definition 5.3. We write $\equiv_!$ for the least congruence relation on processes which results from extending structural congruence \equiv (Definition 2.2) with axioms (1)–(3) below:

1. $(\nu u)(!u(z).P \mid (\nu y)(Q \mid R)) \equiv_! (\nu y)((\nu u)(!u(z).P \mid Q) \mid (\nu u)(!u(z).P \mid R))$
2. $(\nu u)(!u(y).P \mid (\nu v)(!v(z).Q \mid R)) \equiv_! (\nu v)((\nu z).(\nu u)(!u(y).P \mid Q)) \mid (\nu u)(!u(y).P \mid R)$
3. $(\nu u)(!u(y).Q \mid P) \equiv_! P$ if $u \notin \text{fn}(P)$.

Notice that $\equiv_!$ was defined in [11, Definition 4.3], and noted \simeq_s . These axioms are called the *sharpened replication axioms* [42] and are known to express sound behavioral equivalences up to strong bisimilarity in our typed setting.

Some intuitions on Definition 5.3 follow. Axioms (1) and (2) represent principles for the distribution of shared servers among processes, while (3) formalizes the garbage collection of shared servers which cannot be invoked by any process.

Proposition 5.1. Let P and Q be well-typed processes.

1. If $P \rightarrow P'$ and $P \equiv_! Q$ then there is Q' such that $Q \rightarrow Q'$ and $P' \equiv_! Q'$.
2. If $P \xrightarrow{\alpha} P'$ and $P \equiv_! Q$ then there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv_! Q'$.

Proof. By induction on the derivation of $P \equiv_! Q$, then by case analysis on \rightarrow and $\xrightarrow{\alpha}$, respectively. \square

Proposition 5.2. If $P \Downarrow$ and $P \equiv_! Q$ then $Q \Downarrow$.

Proof. Follows by Proposition 5.1, by noticing that: (i) axioms (1) and (2) of $\equiv_!$ do not add new input-guarded replicated processes and (ii) axiom (3) may add a new input-guarded replicated process (if read from right to left) which cannot be invoked. \square

5.2. Logical relations for strong normalization of well-typed processes

We now introduce a theory of linear logical relations for session-typed processes, and use it to prove strong normalization (Definition 5.1).

First step: the logical predicate and its closure properties We define a logical predicate on well-typed processes and establish a few associated closure properties. More precisely, we define a sequent-indexed family of sets of processes (process predicates) so that a set of processes $\mathcal{L}[\Gamma; \Delta \vdash T]$ enjoying certain closure properties is assigned to any sequent $\Gamma; \Delta \vdash T$. The logical predicate is defined by induction on the structure of sequents. The base case, given below, considers sequents with empty left-hand side typing, where we abbreviate $\mathcal{L}[\Gamma; \Delta \vdash T]$ by $\mathcal{L}[T]$.

Definition 5.4 (Logical predicate for termination, base case). For any $T = z:A$ we inductively define $\mathcal{L}[T]$ as the set of all P such that $P \Downarrow$ and $\cdot \vdash P :: T$ and

$$\begin{aligned}
 P \in \mathcal{L}[z:1] & \text{ iff } \forall P'. (P \Rightarrow P' \wedge P' \not\rightarrow) \text{ implies } P' \equiv_! \mathbf{0} \\
 P \in \mathcal{L}[z:A \multimap B] & \text{ iff } \forall P', y. (P \xrightarrow{z(y)} P') \text{ implies } \forall Q \in \mathcal{L}[y:A]. (\nu y)(P' \mid Q) \in \mathcal{L}[z:B] \\
 P \in \mathcal{L}[z:A \otimes B] & \text{ iff } \forall P', y. (P \xrightarrow{z(y)} P') \text{ implies } \exists P_1, P_2. (P' \equiv_! P_1 \mid P_2 \wedge P_1 \in \mathcal{L}[y:A] \wedge P_2 \in \mathcal{L}[z:B]) \\
 P \in \mathcal{L}[z:!A] & \text{ iff } \forall P'. (P \Rightarrow P') \text{ implies } \exists P_1. (P' \equiv_! !z(y).P_1 \wedge P_1 \in \mathcal{L}[y:A]) \\
 P \in \mathcal{L}[z:A \& B] & \text{ iff } (\forall P'. (P \xrightarrow{z.\text{inl}} P') \text{ implies } P' \in \mathcal{L}[z:A]) \wedge (\forall P'. (P \xrightarrow{z.\text{inr}} P') \text{ implies } P' \in \mathcal{L}[z:B]) \\
 P \in \mathcal{L}[z:A \oplus B] & \text{ iff } (\forall P'. (P \xrightarrow{z.\text{inl}} P') \text{ implies } P' \in \mathcal{L}[z:A]) \wedge (\forall P'. (P \xrightarrow{z.\text{inr}} P') \text{ implies } P' \in \mathcal{L}[z:B]).
 \end{aligned}$$

Some comments are in order. First, observe how the definition of $\mathcal{L}[T]$ relies on both reductions and weak transitions, and the fact that processes in the logical predicate are terminating by definition. Also, notice that the use of \equiv_i in $\mathcal{L}[z:\mathbf{1}]$ is justified by the fact that a terminated process may be well the composition of a number of shared servers with no potential clients. Using suitable processes that “close” the derivative of the transition, in $\mathcal{L}[z:A \multimap B]$ and $\mathcal{L}[z:A \otimes B]$ we adhere to the linear logic interpretations for input and output types, respectively. In particular, in $\mathcal{L}[z:A \otimes B]$ it is worth observing how \equiv_i is used to “split” the derivative of the transition, thus preserving consistency with the separate, non-interfering nature of the multiplicative conjunction. The definition of $\mathcal{L}[z:A]$ is also rather structural, relying again on the distribution principles embodied in \equiv_i . The definitions of $\mathcal{L}[z:A \& B]$ and $\mathcal{L}[z:A \oplus B]$ are self-explanatory.

Below, we extend the logical predicate to arbitrary typing environments. Observe how we adhere to the principles of rules (Tcut) and (Tcut!) for this purpose.

Definition 5.5 (Logical predicate for termination, inductive case). For any sequent $\Gamma; \Delta \vdash T$ with a non-empty left-hand side environment, we define $\mathcal{L}[\Gamma; \Delta \vdash T]$ as the set of processes inductively defined as follows:

$$\begin{aligned} P \in \mathcal{L}[\Gamma; y:A, \Delta \vdash T] & \text{ iff } \forall R \in \mathcal{L}[y:A]. (\nu y)(R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T] \\ P \in \mathcal{L}[u:A, \Gamma; \Delta \vdash T] & \text{ iff } \forall R \in \mathcal{L}[y:A]. (\nu u)(!u(y).R \mid P) \in \mathcal{L}[\Gamma; \Delta \vdash T]. \end{aligned}$$

We often rely on the following alternative characterization of the sets $\mathcal{L}[\Gamma; \Delta \vdash T]$.

Definition 5.6. Let $\Gamma = \{u_i:B_i\}_{i \in I}$ and $\Delta = \{x_j:A_j\}_{j \in J}$ be a non-linear and a linear typing environment, respectively. We define the sets of processes \mathcal{C}_Γ and \mathcal{C}_Δ as:

$$\mathcal{C}_\Gamma \stackrel{\text{def}}{=} \left\{ \prod_{i \in I} !u_i(y_i).R_i \mid R_i \in \mathcal{L}[y_i:B_i] \right\} \quad \mathcal{C}_\Delta \stackrel{\text{def}}{=} \left\{ \prod_{j \in J} Q_j \mid Q_j \in \mathcal{L}[x_j:A_j] \right\}.$$

Because of the role of left-hand side typing environments, processes in \mathcal{C}_Γ and \mathcal{C}_Δ are then *logical representatives* of the behavior specified by Γ and Δ , respectively.

Proposition 5.3. Let Γ and Δ be a non-linear and a linear typing environment, resp. Then, for all $Q \in \mathcal{C}_\Gamma$ and for all $R \in \mathcal{C}_\Delta$, we have $Q \Downarrow$ and $R \Downarrow$. Moreover, $Q \not\rightarrow$.

Proof. By Definition 5.6, every process in \mathcal{C}_Δ corresponds to the composition of non-interfering, terminating processes. Hence, $R \Downarrow$. The same applies for processes in \mathcal{C}_Γ , which, by construction, correspond to the composition of input-guarded replicated processes. Hence, $Q \Downarrow$, $Q \not\rightarrow$. \square

Lemma 5.1. Let P be a process such that $\Gamma; \Delta \vdash P :: T$, with $\Gamma = \{u_i:B_i\}_{i \in I}$ and $\Delta = \{x_j:A_j\}_{j \in J}$. We then have:

$$P \in \mathcal{L}[\Gamma; \Delta \vdash T] \text{ iff } \forall Q \in \mathcal{C}_\Gamma, \forall R \in \mathcal{C}_\Delta, (\nu \tilde{u}, \tilde{x})(P \mid Q \mid R) \in \mathcal{L}[T].$$

Proof. Immediate from Definitions 5.5 and 5.6. \square

The following closure properties will be fundamental in the second step of the proof, when we will show that well-typed processes are in the logical predicate. We first state closure of $\mathcal{L}[T]$ with respect to substitution and structural congruence:

Proposition 5.4. Let A be a type. If $P \in \mathcal{L}[z:A]$ then $P\{x/z\} \in \mathcal{L}[x:A]$.

Proof. Immediate from Definition 5.4. \square

Proposition 5.5. Let P, Q be well-typed. If $P \in \mathcal{L}[T]$ and $P \equiv Q$ then $Q \in \mathcal{L}[T]$.

Proof. By induction on the definition of $P \equiv Q$, using Propositions 5.1 and 5.2, and the fact that well-typed processes are closed under \equiv by definition. \square

The next proposition provides a basic liveness guarantee for typed processes.

Proposition 5.6. Let $z; \vdash P :: z:T$ and $P \Downarrow$, with $T \in \{A \otimes B, A \multimap B, A \oplus B, A \& B\}$. Then, there exist α, P' such that $P \xrightarrow{\alpha} P'$, and one of the following holds:

- if $T = A \otimes B$ then $\alpha = \bar{z}(y)$;
- if $T = A \multimap B$ then $\alpha = z(y)$;
- if $T = A \oplus B$ then $\alpha = \bar{z}.\text{inr}$ or $\alpha = \bar{z}.\text{inl}$;
- if $T = A \& B$ then $\alpha = z.\text{inr}$ or $\alpha = z.\text{inl}$.

Proof. Since $T \notin \{1, !T'\}$ then, using Lemma 3.4, we know that $\text{live}(P)$ holds. Hence, Lemma 3.1 can be used to infer that either $P \longrightarrow P'$ or $P \xrightarrow{\alpha} P'$, with $s(\alpha) = z$. Termination ensures that such reductions, before or after α , are finite. This gives us Part (i). Part (ii) on the actual shape of α can be inferred using Lemma 3.2. \square

We now extend Proposition 5.5 so as to state closure of $\mathcal{L}[T]$ under \equiv_1 .

Proposition 5.7. *Let P, Q be well-typed. If $P \in \mathcal{L}[T]$ and $P \equiv_1 Q$ then $Q \in \mathcal{L}[T]$.*

Proof. By induction on the definition of $P \equiv_1 Q$. See Appendix A.1, Page 279. \square

The following property gives us a basic determinacy property for well-typed processes. Below, we write $P \longrightarrow_x P'$ for the reduction step $P \longrightarrow P'$ which results from a synchronization on name x , private to P .

Proposition 5.8. *Let $\cdot; \cdot \vdash P :: z:A$ be a well-typed process. If $P \longrightarrow_x P_1$ and $P \longrightarrow_y P_2$ and $P_1 \neq P_2$ then there exist P'_1, P'_2 such that $P_1 \longrightarrow_y P'_1$ and $P_2 \longrightarrow_x P'_2$.*

Proof. By a case analysis on the different ways in which two different reductions on private names can arise from process P . The last applied rule in both reductions is the contextual rule for name restriction; we then analyze all possible combinations for premises of that rule (input/output, branching/selection, shared server invocation, forwarding), using subject reduction (Theorem 3.1) and progress (Theorem 3.2). See Appendix A.2 (Page 280) for details. \square

Proposition 5.9. *Let $\cdot; \cdot \vdash P :: T$ be a well-typed process. If $P \xRightarrow{\alpha} P^*$ and $P \longrightarrow P'$ then $P' \xRightarrow{\alpha} P_2$, for some P_2 .*

Proof. By definition of weak transition, the assumption $P \xRightarrow{\alpha} P_1$ can be alternatively expressed as $P \Longrightarrow P_0 \xrightarrow{\alpha} P_1 \Longrightarrow P^*$, for some P_0, P_1 . There are two main cases, depending on whether $P \longrightarrow P' \Longrightarrow P_0$ holds, that is, on whether the sequence of reductions from P to P_0 includes P' . If P' is included then the thesis follows trivially. If P' is not included then we have that both $P \longrightarrow P'$ and $P \Longrightarrow P_0$ are enabled. The thesis then follows by Proposition 5.8, which ensures that two enabled reductions from the same process do not preclude each other. \square

We now state *forward* and *backward* closure of the logical predicate with respect to reduction; these are typical ingredients in the method of logical relations.

Proposition 5.10 (Forward closure). *If $P \in \mathcal{L}[T]$ and $P \longrightarrow P'$ then $P' \in \mathcal{L}[T]$.*

Proof. By induction on the structure of T . In all cases, first we must show two conditions: (i) $P' \Downarrow$ and (ii) P' is well-typed. First, by assumption and Definition 5.4, we have that $P \Downarrow$; then, since $P \longrightarrow P'$, we have $P' \Downarrow$ as well. Second, by assumption and Definition 5.4, we have that $\cdot; \cdot \vdash P :: T$, then, by Theorem 3.1, we infer $\cdot; \cdot \vdash P' :: T$. Then, with these two conditions and the assumption $P \longrightarrow P'$, membership of P' in $\mathcal{L}[T]$ is established depending on the structure of T . If $T = z:1$ or $T = z:!A$ then one uses Proposition 5.8 to show that if $P \Longrightarrow P_1$ then also $P' \Longrightarrow P_2$, for some P_2 . In the other cases one uses by Proposition 5.9 to show that if $P \xRightarrow{\alpha} P_1$ then also $P' \xRightarrow{\alpha} P_2$, for some P_2 . \square

Proposition 5.11 (Backward closure). *Let $\cdot; \cdot \vdash P :: T$ be a well-typed process. If for all P_i such that $P \longrightarrow P_i$ we have that $P_i \in \mathcal{L}[T]$ implies $P \in \mathcal{L}[T]$.*

Proof. By induction on the structure of T . In all cases, we must show: (i) $P \Downarrow$, (ii) P is well-typed, and (iii) $P \in \mathcal{L}[T]$, as in Definition 5.4. First, by assumption we have $P_i \Downarrow$ for every reduction P_i of P ; then, since $P \longrightarrow P_i$, we have $P \Downarrow$ directly from Definition 5.1. Second, well-typedness of P is given by assumption. For (iii), we proceed depending on the structure of T , exploiting the fact that $P_i \in \mathcal{L}[T]$. If $T = z:1$ then Definition 5.4 ensures that $\forall P'. (P_i \Longrightarrow P' \wedge P' \not\rightarrow) \implies P' \equiv_1 0$. Now, since by assumption we have $P \longrightarrow P_i$, the definition of weak transition ensures that $P \longrightarrow P_i$ and $P_i \Longrightarrow P'$ imply $P \Longrightarrow P'$, for any P' . Therefore, $P \in \mathcal{L}[z:1]$. If $T = z:!A$ the reasoning is similar, for membership in $\mathcal{L}[T]$ also relies on an unlabeled weak transition. In all other cases, membership in $\mathcal{L}[T]$ depends on a labeled weak transition: Definition 5.4 ensures that every P' such that $P_i \xRightarrow{\alpha} P'$ satisfies some condition—the exact shape of α is determined using Proposition 5.6. Now, since by assumption we have $P \longrightarrow P_i$, definition of weak transition ensures that $P \longrightarrow P_i$ and $P_i \xRightarrow{\alpha} P'$ imply $P \xRightarrow{\alpha} P'$, for any P' . Thus, $P \in \mathcal{L}[T]$. \square

The final closure property concerns process composition:

Proposition 5.12. *Let P, Q be processes such that $P \in \mathcal{L}[T]$ and $Q \in \mathcal{L}[-:1]$. Then, $P \mid Q \in \mathcal{L}[T]$.*

Proof. By induction on the structure of T . See Appendix A.3, Page 283. \square

Second step: well-typed processes are in the logical predicate We now prove that well-typed processes are in the logical predicate. Because of Definition 5.4, termination of well-typed processes will follow as a consequence.

Lemma 5.2. *Let P be a process. If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$.*

Proof. By induction on the derivation of $\Gamma; \Delta \vdash P :: T$, with a case analysis on the last typing rule used. We have 18 cases to check; in all of them, we use Lemma 5.1 to show that every $M = (\nu \tilde{u}, \tilde{x})(P \mid G \mid D)$ with $G \in \mathcal{C}_\Gamma$ and $D \in \mathcal{C}_\Delta$, is in $\mathcal{L}[T]$. In case (Tid), we use Proposition 5.4 (closure wrt substitution) and Proposition 5.11 (backward closure). In cases (T \otimes L), (T \multimap L), (Tcopy), (T \oplus L), (T $\&$ L₁), and (T $\&$ L₂), we proceed in two steps: first, using Proposition 5.10 (forward closure) we show that every M'' such that $M \Rightarrow M''$ is in $\mathcal{L}[T]$; then, we combine this result with Proposition 5.11 (backward closure) to conclude that $M \in \mathcal{L}[T]$. In cases (T1R), (T \otimes R), (T \multimap R), (T!R), (T \oplus R₁), and (T \oplus R₂), we show that M conforms to a specific case of Definition 5.4. Case (T1L) uses Proposition 5.12. Cases (T \otimes L), (T \multimap L), (T \oplus L), and (T $\&$ L₁) use the liveness guarantee given by Proposition 5.6. Cases (Tcopy), (T!L), and (Tcut^l) use Proposition 5.5 (closure under \equiv). Cases (Tcut), (T \multimap R), and (T!R) use Proposition 5.7 (closure under \equiv). See Appendix A.4, Page 279 for details. \square

We now state our first main result: well-typed processes terminate.

Theorem 5.1 (Well-typed processes are terminating). *If $\Gamma; \Delta \vdash P :: T$ then $P \Downarrow$.*

Proof. Follows from previously proven facts:

$\Gamma; \Delta \vdash P :: T$	[Assumption]	(a)
$P \in \mathcal{L}[\Gamma; \Delta \vdash T]$	[By Lemma 5.2 and (a)]	(b)
Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:		
$G \Downarrow, D \Downarrow$	[By Proposition 5.3]	(c)
$(\nu \tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}[T]$	[By Lemma 5.1 on (b)]	(d)
$(\nu \tilde{u}, \tilde{x})(P \mid G \mid D) \Downarrow$	[From (d) and Definition 5.4]	(e)
$P \Downarrow$	[Consequence of (c) and (e)].	\square

5.3. Logical relations for confluence of well-typed processes

We now adapt the logical relations and the proof technique of Section 5.2 to the case of *confluence* (Definition 5.2).

The proof that well-typed processes are confluent is very similar to that of termination; required adjustments concern mainly closure properties.

Proposition 5.13 (Properties of confluent processes). *Assume well-typed processes $P, P', P_1, \dots, P_k, Q$.*

1. *Forward closure: If $P \Diamond$ and $P \longrightarrow P'$ then $P' \Diamond$.*
2. *Backward closure: If for all P_i such that $P \longrightarrow P_i$ we have that $P_i \Diamond$, then $P \Diamond$.*
3. *Closure wrt composition: Let P, Q be such that (i) $\cdot \vdash P :: x:A$, (ii) $\cdot \vdash x:A \vdash Q :: T$, (iii) $P \Diamond$, and (iv) $Q \Diamond$. Then $(\nu x)(P \mid Q) \Diamond$.*

Proof. See Appendix A.5 (Page 291) for details. \square

Proposition 5.14. *If $P \Diamond$ and $P \equiv_i Q$ then $Q \Diamond$.*

Proof. Follows immediately from Proposition 5.1. \square

First step: the logical predicate and its closure properties The logical predicate for confluence, given below, is essentially the same as the one in [Definition 5.4](#). Hence, subsequent auxiliary definitions and closure properties mirror those in [Section 5.2](#).

Definition 5.7 (Logical predicate for confluence, base case). For any $T = z:A$ we inductively define $\mathcal{L}^\diamond[T]$ as the set of all P such that $P \diamond$ and $\cdot; \vdash P :: T$ and

$$\begin{aligned} P \in \mathcal{L}^\diamond[z:1] & \text{ iff } \forall P'. (P \implies P' \wedge P' \not\rightarrow) \text{ implies } P' \equiv! 0 \\ P \in \mathcal{L}^\diamond[z:A \multimap B] & \text{ iff } \forall P' y. (P \xrightarrow{z(y)} P') \text{ implies } \forall Q \in \mathcal{L}^\diamond[y:A]. (\nu y)(P' \mid Q) \in \mathcal{L}^\diamond[z:B] \\ P \in \mathcal{L}^\diamond[z:A \otimes B] & \text{ iff } \forall P' y. (P \xrightarrow{\bar{z}(y)} P') \text{ implies } \exists P_1, P_2. (P' \equiv! P_1 \mid P_2 \wedge P_1 \in \mathcal{L}^\diamond[y:A] \wedge P_2 \in \mathcal{L}^\diamond[z:B]) \\ P \in \mathcal{L}^\diamond[z;!A] & \text{ iff } \forall P'. (P \implies P') \text{ implies } \exists P_1. (P' \equiv! z(y).P_1 \wedge P_1 \in \mathcal{L}^\diamond[y:A]) \\ P \in \mathcal{L}^\diamond[z:A \& B] & \text{ iff } (\forall P'. (P \xrightarrow{z.\text{inl}} P') \text{ implies } P' \in \mathcal{L}^\diamond[z:A]) \wedge (\forall P'. (P \xrightarrow{z.\text{inr}} P') \text{ implies } P' \in \mathcal{L}^\diamond[z:B]) \\ P \in \mathcal{L}^\diamond[z:A \oplus B] & \text{ iff } (\forall P'. (P \xrightarrow{\bar{z}.\text{inl}} P') \text{ implies } P' \in \mathcal{L}^\diamond[z:A]) \wedge (\forall P'. (P \xrightarrow{\bar{z}.\text{inr}} P') \text{ implies } P' \in \mathcal{L}^\diamond[z:B]). \end{aligned}$$

Below, we extend $\mathcal{L}^\diamond[T]$ to arbitrary typing environments.

Definition 5.8 (Logical predicate–inductive case). For any sequent $\Gamma; \Delta \vdash T$ with a non-empty left hand side environment, we define $\mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$ to be the set of processes inductively defined as follows:

$$\begin{aligned} P \in \mathcal{L}^\diamond[\Gamma; y:A, \Delta \vdash T] & \text{ if } \forall R \in \mathcal{L}^\diamond[y:A]. (\nu y)(R \mid P) \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T] \\ P \in \mathcal{L}^\diamond[u:A, \Gamma; \Delta \vdash T] & \text{ if } \forall R \in \mathcal{L}^\diamond[y:A]. (\nu u)(!u(y).R \mid P) \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]. \end{aligned}$$

We often rely on the following characterization of the sets $\mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$.

Definition 5.9. Let $\Gamma = \{u_i:B_i\}_{i \in I}$ and $\Delta = \{x_j:A_j\}_{j \in J}$ be a non-linear and a linear typing environment, respectively. We define the sets of processes $\mathcal{C}_\Gamma^\diamond$ and $\mathcal{C}_\Delta^\diamond$ as:

$$\mathcal{C}_\Gamma^\diamond \stackrel{\text{def}}{=} \left\{ \prod_{i \in I} !u_i(y_i).R_i \mid R_i \in \mathcal{L}^\diamond[y_i:B_i] \right\} \quad \mathcal{C}_\Delta^\diamond \stackrel{\text{def}}{=} \left\{ \prod_{j \in J} Q_j \mid Q_j \in \mathcal{L}^\diamond[x_j:A_j] \right\}.$$

We define sets of processes $\mathcal{C}_\Gamma^\diamond$ and $\mathcal{C}_\Delta^\diamond$ as logical representatives of the behavior specified by Γ and Δ , respectively.

Proposition 5.15. Let Γ and Δ be a non-linear and a linear typing environment, respectively. Then, for all $Q \in \mathcal{C}_\Gamma^\diamond$ and for all $R \in \mathcal{C}_\Delta^\diamond$, we have $Q \diamond$ and $R \diamond$. Furthermore, $Q \not\rightarrow$ and $R \Downarrow$.

Proof. By [Definition 5.9](#), every $R \in \mathcal{C}_\Delta^\diamond$ corresponds to the composition of independent, confluent processes. Hence, using [Proposition 5.13](#) (3), we have $R \diamond$. Also, R is the composition of well-typed processes, which by [Theorem 5.1](#) are all terminating. Hence, $R \Downarrow$. As for $Q \in \mathcal{C}_\Gamma^\diamond$, by construction it corresponds to the composition of input-guarded replicated processes. Hence, $Q \not\rightarrow$. \square

Lemma 5.3. Let $\Gamma; \Delta \vdash P :: T$, with $\Gamma = \{u_i:B_i\}_{i \in I}$ and $\Delta = \{x_j:A_j\}_{j \in J}$. We have: $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$ iff $\forall Q \in \mathcal{C}_\Gamma^\diamond, \forall R \in \mathcal{C}_\Delta^\diamond, (\nu \tilde{u}, \tilde{x})(P \mid Q \mid R) \in \mathcal{L}^\diamond[T]$.

Proof. The proof follows from [Definitions 5.8 and 5.9](#), [Proposition 5.15](#), and closure of confluent processes under composition ([Proposition 5.13\(3\)](#)). \square

We now state the closure properties required to show that well-typed processes are in the logical predicate for confluence.

Proposition 5.16. Let A be a type. If $P \in \mathcal{L}^\diamond[z:A]$ then $P\{x/z\} \in \mathcal{L}^\diamond[x:A]$.

Proof. Immediate from [Definition 5.7](#). \square

Proposition 5.17. Let P, Q be well-typed. If $P \in \mathcal{L}^\diamond[T]$ and $P \equiv Q$ then $Q \in \mathcal{L}^\diamond[T]$.

Proof. By induction on the definition of $P \equiv Q$, using [Propositions 5.1 and 5.14](#), and the fact that well-typed processes are closed under \equiv by definition. \square

We now extend [Proposition 5.17](#) so as to state closure of $\mathcal{L}^\diamond[T]$ under \equiv .

Proposition 5.18. *Let P, Q be well-typed. If $P \in \mathcal{L}^\diamond[T]$ and $P \equiv Q$ then $Q \in \mathcal{L}^\diamond[T]$.*

Proof. By induction on the definition of $P \equiv Q$, following the lines of the proof of [Proposition 5.7](#). \square

We now state *forward* and *backward* closure of $\mathcal{L}^\diamond[T]$ with respect to reduction.

Proposition 5.19 (Forward closure). *If $P \in \mathcal{L}^\diamond[T]$ and $P \longrightarrow P'$ then $P' \in \mathcal{L}^\diamond[T]$.*

Proof. By induction on the structure of T . In all cases, we must show two conditions: (i) $P' \diamond$ and (ii) P' is well-typed. Observe that by assumption and [Definition 5.7](#) we have both $P \diamond$ and $\cdot; \cdot \vdash P :: T$. First, since $P \diamond$ and $P \longrightarrow P'$, by [Proposition 5.13](#) (1) we infer $P' \diamond$. Second, since $\cdot; \cdot \vdash P :: T$, [Theorem 3.1](#) ensures $\cdot; \cdot \vdash P' :: T$. Then, with these two conditions and the assumption $P \longrightarrow P'$, membership of P' in $\mathcal{L}^\diamond[T]$ is established depending on the structure of T . If $T = z:1$ or $T = z:A$ then one uses [Proposition 5.8](#) to show that if $P \Longrightarrow P_1$ then also $P' \Longrightarrow P_2$, for some P_2 . In the other cases one uses by [Proposition 5.9](#) to show that if $P \xRightarrow{\alpha} P_1$ then also $P' \xRightarrow{\alpha} P_2$, for some P_2 . \square

The proof of the following proposition follows closely that of [Proposition 5.11](#):

Proposition 5.20 (Backward closure). *Let $\cdot; \cdot \vdash P :: T$ be a well-typed process. If for all P_i such that $P \longrightarrow P_i$ we have that $P_i \in \mathcal{L}^\diamond[T]$ implies $P \in \mathcal{L}^\diamond[T]$.*

Proof. By induction on the structure of T . In all cases, we must show: (i) $P \diamond$, (ii) P is well-typed, and (iii) $P \in \mathcal{L}^\diamond[T]$, as in [Definition 5.7](#). First, item (i) follows from [Proposition 5.13](#) (2). Second, well-typedness of P is given by assumption. For (iii), we proceed depending on the structure of T , exploiting the fact that $P_i \in \mathcal{L}^\diamond[T]$. If $T = z:1$ then [Definition 5.7](#) ensures that $\forall P'. (P_i \Longrightarrow P' \wedge P' \not\rightarrow) \implies P' \equiv 0$. Now, since by assumption we have $P \longrightarrow P_i$, the definition of weak transition ensures that $P \longrightarrow P_i$ and $P_i \Longrightarrow P' \implies P \Longrightarrow P'$, for any P' . Therefore, $P \in \mathcal{L}^\diamond[z:1]$. If $T = z:A$ the reasoning is similar, for membership in $\mathcal{L}^\diamond[T]$ also relies on an unlabeled weak transition. In all other cases, membership in $\mathcal{L}^\diamond[T]$ depends on a labeled weak transition: [Definition 5.7](#) ensures that every P' such that $P_i \xRightarrow{\alpha} P'$ satisfies some condition—the exact shape of α is determined using [Proposition 5.6](#). Now, since by assumption we have $P \longrightarrow P_i$, definition of weak transition ensures that $P \longrightarrow P_i$ and $P_i \xRightarrow{\alpha} P' \implies P \xRightarrow{\alpha} P'$, for any P' . Thus, $P \in \mathcal{L}^\diamond[T]$. \square

Second step: well-typed processes are in the logical predicate We now prove that well-typed processes are in the logical predicate.

Lemma 5.4. *Let P be a process. If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$.*

Proof. By induction on the derivation of $\Gamma; \Delta \vdash P :: T$, with a case analysis on the last typing rule used. See [Appendix A.6](#) (Page 291) for details. \square

We now state the desired result: well-typed processes are confluent.

Theorem 5.2 (Well-typed processes are confluent). *If $\Gamma; \Delta \vdash P :: T$ then $P \diamond$.*

Proof. Follows from previously proven facts. By assumption, we have $\Gamma; \Delta \vdash P :: T$. Using this and [Lemma 5.4](#) we get $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$. Pick any $G \in \mathcal{C}_P^\diamond, D \in \mathcal{C}_\Delta^\diamond$: combining $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$ and [Lemma 5.3](#) gives us $(\nu \tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}^\diamond[T]$. By using this, together with [Definition 5.7](#), we infer $(\nu \tilde{u}, \tilde{x})(P \mid G \mid D) \diamond$. Since [Proposition 5.15](#) ensures $G \diamond$ and $D \diamond$, this last result implies $P \diamond$. \square

6. Observational equivalences for session-typed processes

In this section, we investigate the behavioral theory for session-typed processes. We introduce *typed context bisimilarity* (noted \approx), a labelled bisimulation which closely follows the nature of typing judgments.

6.1. Auxiliary definitions

Recall that $\vdash P :: T$ stands for $\cdot; \cdot \vdash P :: T$. We sometimes write $\Gamma; \Delta \vdash P, Q :: T$ to mean that both $\Gamma; \Delta \vdash P :: T$ and $\Gamma; \Delta \vdash Q :: T$ hold. Below, we use S to range over sequents of the form $\Gamma; \Delta \vdash T$. We will rely on *type-respecting* relations, which are indexed by sequents S . We will use binary relations, so the adjective “binary” will be always omitted.

Definition 6.1 (*Type-respecting relations*). A *type-respecting relation* over processes, written $\{\mathcal{R}_S\}_S$, is defined as a family of relations over processes indexed by S . We often write \mathcal{R} to refer to the whole family, and use notation $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$ to mean both (i) $\Gamma; \Delta \vdash P, Q :: T$ and (ii) $(P, Q) \in \mathcal{R}_{\Gamma; \Delta \vdash T}$.

We use $\mathcal{R}, \mathcal{R}', \dots$ to range over type-respecting relations. In the following, we will often omit the adjective “type-respecting”.

Definition 6.2. A relation \mathcal{R} is said to be

- *Reflexive*, if $\Gamma; \Delta \vdash P :: T$ implies $\Gamma; \Delta \vdash P \mathcal{R} P :: T$;
- *Symmetric*, if $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$ implies $\Gamma; \Delta \vdash Q \mathcal{R} P :: T$;
- *Transitive*, $\Gamma; \Delta \vdash P \mathcal{R} P' :: T$ and $\Gamma; \Delta \vdash P' \mathcal{R} Q :: T$ imply $\Gamma; \Delta \vdash P \mathcal{R} Q :: T$.

Moreover, \mathcal{R} is said to be an *equivalence* if it is reflexive, symmetric, and transitive.

In order to define *contextual* relations, we introduce a natural notion of (*typed*) *process contexts*. Intuitively, a context is a process that contains one hole, noted \bullet . Holes are *typed*: a hole, denoted by $\bullet_{\Gamma; \Delta \vdash T}$, can only be filled in with a process matching its type. We shall use K, K', \dots for ranging over properly defined contexts, in the sense given next. We rely on left- and right-hand side typings for defining contexts and its properties precisely. We consider contexts with exactly one hole, but our definitions are easy to generalize.

We rely on a minimal extension of the syntax of processes (Definition 2.1) with \bullet . We then extend sequents, in the following way:

$$\mathcal{H}; \Gamma; \Delta \vdash K :: z:C$$

\mathcal{H} specifies the typing requirements of the unique hole occurring in (context) K , and is thus always of the form $\bullet_{\Gamma; \Delta \vdash T}$ for some Γ, Δ and T : we have that

$$\bullet_{\Gamma; \Delta \vdash T}; \Gamma; \Delta' \vdash K :: z:C$$

is the type of a context K whose hole is to be substituted by some process P such that $\Gamma; \Delta \vdash P :: T$. As a result of the substitution, we obtain a process $\Gamma; \Delta' \vdash K[P] :: z:C$. Since we consider at most one hole, \mathcal{H} is either empty or has exactly one element. If \mathcal{H} is empty then K is a process and we obtain the usual typing rules; we write $\Gamma; \Delta \vdash R :: T$ rather than $\cdot; \Gamma; \Delta \vdash R :: T$. The definition of typed contexts is completed by extending the type system with the following two rules:

$$\begin{array}{c} \text{(Thole)} \\ \hline \bullet_{\Gamma; \Delta \vdash T}; \Gamma; \Delta \vdash \bullet :: T \end{array} \quad \begin{array}{c} \text{(Tfill)} \\ \hline \frac{\Gamma; \Delta \vdash R :: T \quad \bullet_{\Gamma; \Delta \vdash T}; \Gamma; \Delta' \vdash K :: z:C}{\Gamma; \Delta' \vdash K[R] :: z:C} \end{array}$$

Axiom (Thole) allows to introduce holes into typed contexts. In rule (Tfill), R is a process (it does not have any holes), and K is a context with a hole of type $\Gamma; \Delta \vdash T$. The substitution of occurrences of \bullet in K with R , noted $K[R]$ is sound as long as the typings of R coincide with those declared in \mathcal{H} for K .

As an example, consider a simple parallel context, $(\nu x)(\bullet \mid P)$ which is filled in with an appropriately typed process R :

$$\frac{\begin{array}{c} \vdots \\ \hline \Gamma; \Delta_1 \vdash P :: x:C \end{array} \quad \frac{\bullet_{\Gamma; x:C, \Delta_2 \vdash T}; \Gamma; x:C, \Delta_2 \vdash \bullet :: T}{\bullet_{\Gamma; x:C, \Delta_2 \vdash T}; \Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P \mid \bullet) :: T} \text{(Thole)}}{\Gamma; x:C, \Delta_2 \vdash R :: T \quad \bullet_{\Gamma; x:C, \Delta_2 \vdash T}; \Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P \mid \bullet) :: T} \text{(Tcut)} \quad \text{(Tfill)}$$

As we have seen, contexts in our setting are hardly arbitrary: only type-compatible processes are inserted into holes. Based on this observation, and following the typing rules, we define a notion of contextual relation in our typed setting:

Definition 6.3 (*Contextual relation*). A relation \mathcal{R} is *contextual* if it satisfies the conditions in Fig. 5 (Page 271).

A type-respecting relation \mathcal{R} is contextual if

0. $\Gamma; \Delta \vdash P \mathcal{R} Q :: y:A \text{ implies } \Gamma; \Delta \vdash (\nu y)(P \mid [y \leftrightarrow z]) \mathcal{R} (\nu y)(Q \mid [y \leftrightarrow z]) :: z:A$, for any z such that $\Gamma; y:A \vdash [y \leftrightarrow z] :: z:A$
1. $\Gamma; \Delta, y:A \vdash P \mathcal{R} Q :: x:B \text{ implies } \Gamma; \Delta \vdash x(y).P \mathcal{R} x(y).Q :: x:A \multimap B$
2. $\Gamma; \Delta \vdash P \mathcal{R} Q :: y:A \text{ implies } \Gamma; \Delta, \Delta' \vdash \tilde{x}(y).(P \mid S) \mathcal{R} \tilde{x}(y).(Q \mid S) :: x:A \otimes B$, for any x, S, B, Δ' such that $\Gamma; \Delta' \vdash S :: x:B$
3. $\Gamma; \Delta' \vdash P \mathcal{R} Q :: x:B \text{ implies } \Gamma; \Delta, \Delta' \vdash \tilde{x}(y).(S \mid P) \mathcal{R} \tilde{x}(y).(S \mid Q) :: x:A \otimes B$, for any y, S, A, Δ' such that $\Gamma; \Delta \vdash S :: y:A$
4. $\Gamma; \Delta \vdash P \mathcal{R} Q :: x:A \text{ implies } \Gamma; \Delta \vdash x.\text{case}(P, S) \mathcal{R} x.\text{case}(Q, S) :: x:A \& B$, for any S, B such that $\Gamma; \Delta \vdash S :: x:B$
5. $\Gamma; \Delta \vdash P \mathcal{R} Q :: x:B \text{ implies } \Gamma; \Delta \vdash x.\text{case}(S, P) \mathcal{R} x.\text{case}(S, Q) :: x:A \& B$, for any S, A such that $\Gamma; \Delta \vdash S :: x:A$
6. $\Gamma; \Delta \vdash P \mathcal{R} Q :: x:A \text{ implies } \Gamma; \Delta \vdash x.\text{in}x; P \mathcal{R} x.\text{in}x; Q :: x:A \oplus B$, for any B
7. $\Gamma; \Delta \vdash P \mathcal{R} Q :: x:A \text{ implies } \Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid S) \mathcal{R} (\nu x)(Q \mid S) :: T$, for any S, T, Δ' such that $\Gamma; \Delta', x:A \vdash S :: T$
8. $\Gamma; \Delta, x:A \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, \Delta' \vdash (\nu x)(S \mid P) \mathcal{R} (\nu x)(S \mid Q) :: T$, for any S, Δ' such that $\Gamma; \Delta' \vdash S :: x:A$
9. $\Gamma; \cdot \vdash P \mathcal{R} Q :: y:A \text{ implies } \Gamma; \Delta \vdash (\nu u)(!u(y).P \mid S) \mathcal{R} (\nu u)(!u(y).Q \mid S) :: T$, for any u, S, T, Δ such that $\Gamma; u:A; \Delta \vdash S :: T$
10. $\Gamma; u:A; \Delta \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta \vdash (\nu u)(!u(y).S \mid P) \mathcal{R} (\nu u)(!u(y).S \mid Q) :: T$, for any S, y such that $\Gamma; \cdot \vdash S :: y:A$
11. $\Gamma; \Delta \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, \Delta' \vdash S \mid P \mathcal{R} S \mid Q :: T$, for any S, Δ' such that $\Gamma; \Delta' \vdash S :: -1$
12. $\Gamma; \Delta \vdash P \mathcal{R} Q :: -1 \text{ implies } \Gamma; \Delta, \Delta' \vdash P \mid S \mathcal{R} Q \mid S :: T$, for any S, T, Δ' such that $\Gamma; \Delta' \vdash S :: T$
13. $\Gamma; u:A; \Delta \vdash P\{u/x\} \mathcal{R} Q\{u/x\} :: T \text{ implies } \Gamma; \Delta, x:A \vdash P \mathcal{R} Q :: T$
14. $\Gamma; \cdot \vdash P \mathcal{R} Q :: y:A \text{ implies } \Gamma; \cdot \vdash !x(y).P \mathcal{R} !x(y).Q :: x:A!$, for any x
15. $\Gamma; \Delta, y:A, x:B \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, x:A \otimes B \vdash x(y).P \mathcal{R} x(y).Q :: T$
16. $\Gamma; \Delta \vdash P \mathcal{R} Q :: y:A \text{ implies } \Gamma; \Delta, \Delta', x:A \multimap B \vdash \tilde{x}(y).(P \mid S) \mathcal{R} \tilde{x}(y).(Q \mid S) :: T$, for any x, B, S, T, Δ' such that $\Gamma; \Delta', x:B \vdash S :: T$
17. $\Gamma; \Delta, x:B \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, \Delta', x:A \multimap B \vdash \tilde{x}(y).(P \mid S) \mathcal{R} \tilde{x}(y).(Q \mid S) :: T$, for any y, A, S, Δ' such that $\Gamma; \Delta' \vdash S :: y:A$
18. $\Gamma; u:A; \Delta, y:A \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; u:A; \Delta \vdash \bar{u}(y).P \mathcal{R} \bar{u}(y).Q :: T$
19. $\Gamma; \Delta, x:A \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, x:A \oplus B \vdash x.\text{case}(P, S) \mathcal{R} x.\text{case}(Q, S) :: T$, for any x, S, B such that $\Gamma; \Delta, x:B \vdash S :: T$
20. $\Gamma; \Delta, x:B \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, x:A \oplus B \vdash x.\text{case}(S, P) \mathcal{R} x.\text{case}(S, Q) :: T$, for any A, S, T such that $\Gamma; \Delta, x:A \vdash S :: T$
21. $\Gamma; \Delta, x:A \vdash P \mathcal{R} Q :: T \text{ implies } \Gamma; \Delta, x:A \& B \vdash x.\text{in}x; P \mathcal{R} x.\text{in}x; Q :: T$

Fig. 5. Conditions for contextual type-respecting relations (cf. Definition 6.3).

In Fig. 5, we write $x.\text{in}x; P$ to stand for both $x.\text{in}r; P$ and $x.\text{in}l; P$. Some comments to the conditions associated to Definition 6.3 are in order. In all cases, observe how the typing rules guide the shape of allowed contexts. For instance, item (1) is easily seen to correspond to rule (T \multimap R) and associated to the input context

$$\bullet \Gamma; \Delta, y:A \vdash x:B; \Gamma; \Delta \vdash x(y). \bullet :: x:A \multimap B$$

to be filled in by any P such that $\Gamma; \Delta, y:A \vdash P :: x:B$. In fact, premises of each rule suggest where to place holes; rules with two premises lead to two different contexts. Observe how item (0) involves the forwarding construct; this could be seen as a form of closure under substitution, which renames the right-hand side typing of a process. Items (8)–(13) correspond to closure with respect to parallel contexts, which in our typed setting also involves closure with respect to restriction, following rules (Tcut) and (Tcut^l). Notice that while closure under arbitrary process composition is not allowed, closure under independent parallel composition (cf. rule (INDCOMP) in Section 3) is permitted (cf. Items (12) and (13)).

Remark 6.1. Notice that not all the contextuality conditions in Fig. 5 apply in the case \mathcal{R} relates processes related under empty left-hand side typing environments. Indeed, only items (0), (2)–(8), (10)–(13), and (15) apply in that case.

6.2. Typed context bisimilarity

We define *typed context bisimilarity*, a labeled bisimilarity for typed processes. It is defined contextually, as a binary relation indexed over sequents. Roughly, typed context bisimilarity equates two processes if, once coupled with all of their requirements (as described by the left-hand side typing), they perform the same actions (as described by the right-hand side typing). To formalize this intuition, we rely on a combination of inductive and coinductive arguments. The base case of the definition covers the cases in which the left-hand side typing environment is empty (i.e., the process requires nothing from its context to execute): the bisimulation game is then defined by induction on the structure of the (right-hand side) typing, following the expected behavior in each case. The inductive case covers the cases in which the left-hand side typing environment is not empty: the tested processes are put in parallel with processes implementing the required behaviors (as described in the left-hand side typing).

Definition 6.4 (*Typed context bisimilarity*). A symmetric type-respecting binary relation over processes \mathcal{R} is a *typed context bisimulation* if

Base Cases

Tau $\vdash P \mathcal{R} Q :: T$ implies that for all P' such that $P \xrightarrow{\tau} P'$, there exists a Q' such that $Q \Longrightarrow Q'$ and $\vdash P' \mathcal{R} Q' :: T$

Input $\vdash P \mathcal{R} Q :: x:A \multimap B$ implies that for all P' such that $P \xrightarrow{x(y)} P'$, there exists a Q' such that $Q \xrightarrow{x(y)} Q'$ and for all R such that $\vdash R :: y:A, \vdash (\nu y)(R \mid P') \mathcal{R} (\nu y)(R \mid Q') :: x:B$.

Output $\vdash P\mathcal{R}Q :: x:A \otimes B$ implies that for all P' such that $P \xrightarrow{\bar{x}(y)} P'$, there exists a Q' such that $Q \xrightarrow{\bar{x}(y)} Q'$ and for all R such that $\vdash y:A \vdash R :: \neg!1, \vdash (\nu y)(P' | R)\mathcal{R}(\nu y)(Q' | R) :: x:B$.

Replication $\vdash P\mathcal{R}Q :: x:!A$ implies that for all P' such that $P \xrightarrow{x(z)} P'$, there exists a Q' such that $Q \xrightarrow{x(z)} Q'$ and, for all R such that $\vdash y:A \vdash R :: \neg!1, \vdash (\nu z)(P' | R)\mathcal{R}(\nu z)(Q' | R) :: x:!A$.

Choice $\vdash P\mathcal{R}Q :: x:A \& B$ implies both:

- If $P \xrightarrow{x.inl} P'$ then $\vdash P'\mathcal{R}Q' :: x:A$, for some Q' such that $Q \xrightarrow{x.inl} Q'$; and
- If $P \xrightarrow{x.inr} P'$ then $\vdash P'\mathcal{R}Q' :: x:B$, for some Q' such that $Q \xrightarrow{x.inr} Q'$.

Selection $\vdash P\mathcal{R}Q :: x:A \oplus B$ implies both:

- If $P \xrightarrow{x.inl} P'$ then $\vdash P'\mathcal{R}Q' :: x:A$, for some Q' such that $Q \xrightarrow{x.inl} Q'$; and
- If $P \xrightarrow{x.inr} P'$ then $\vdash P'\mathcal{R}Q' :: x:B$, for some Q' such that $Q \xrightarrow{x.inr} Q'$.

Inductive Cases

Linear Names $\Gamma; \Delta, y:A \vdash P\mathcal{R}Q :: T$ implies that for all R such that $\vdash R :: y:A$, then $\Gamma; \Delta \vdash (\nu y)(R | P)\mathcal{R}(\nu y)(R | Q) :: T$.

Shared Names $\Gamma, u:A; \Delta \vdash P\mathcal{R}Q :: T$ implies that for all R such that $\vdash R :: z:A$, then $\Gamma; \Delta \vdash (\nu u)!(u(z).R | P)\mathcal{R}(\nu u)!(u(z).R | Q) :: T$.

We write \approx for the union of all typed context bisimulations, and call it *typed context bisimilarity*.

In all cases, a strong action is matched with a weak transition. In proofs, we shall exploit the fact that [Theorems 5.1 and 5.2](#) ensure that such weak transitions always have finite and confluent reductions. In the base case, the clauses for input, output, and replication decree the closure of the tested processes with a process R that “complements” the continuation of the tested behavior; observe the very similar treatment for output and replication (where R *depends* on some behavior), and contrast it with that for input (where R *provides* the behavior). Also, notice how all clauses but that for replication are defined coinductively for the tested processes (in the sense that closed evolutions should be in the relation), but inductively on the type indexing the relation—the clause for replication may be thus considered as the only fully coinductive one. Also worth noticing is how the closures defined in such clauses (and those defined by the clauses in the inductive case) follow closely the spirit of (Tcut/Tcut') rules in the type system.

6.3. Properties of typed context bisimilarity

We establish some properties of typed context bisimilarity: equivalence ([Proposition 6.1](#)); closure under independent parallel composition ([Proposition 6.2](#)); a simplification for the bisimulation proof technique ([Proposition 6.3](#)); contextuality/congruence ([Lemma 6.1](#)); and τ -inertness ([Lemma 6.2](#)).

Proposition 6.1. \approx is an equivalence, in the sense of [Definition 6.2](#).

Proof. Reflexivity and symmetry are immediate from the definition of type-respecting relations. For transitivity, one shows that for any Γ, Δ, T , relation

$$\mathcal{R} = \{(P, R) \mid \text{there is } Q \text{ with } \Gamma; \Delta \vdash P \approx Q :: T \wedge \Gamma; \Delta \vdash Q \approx R :: T\}$$

is a typed context bisimulation. Suppose $P \xrightarrow{\alpha} P'$; we must find a matching action from R , i.e., $R \xrightarrow{\alpha} R'$. The existence of such an action follows directly from the assumptions $\Gamma; \Delta \vdash P \approx Q :: T$ and $\Gamma; \Delta \vdash Q \approx R :: T$. The reasoning when R moves first is analogous. \square

Proposition 6.2 (Closure under independent composition). Let P, Q, S be processes such that $\Gamma; \Delta \vdash P \approx Q :: T$ and $\Gamma; \Delta' \vdash S :: \neg!1$ hold. Then we have: $\Gamma; \Delta, \Delta' \vdash P | S \approx Q | S :: T$.

Proof. Straightforward by showing the appropriate bisimulation, using the fact that composition with arbitrary processes offering type 1 is type preserving, and by noticing that S cannot interact with P, Q . \square

[Definition 6.4](#) immediately suggests a proof technique for showing that two processes are typed context bisimilar. First, close the processes with *parallel representatives* of their context, applying repeatedly the inductive cases until the left-hand side typing is empty. Then, follow the usual co-inductive proof technique, and show a typed-respecting relation containing the processes obtained in the first step. More precisely, given a left-hand side typing $\Gamma; \Delta$, below we define the set $\mathcal{K}_{\Gamma; \Delta \vdash T}$ of parallel representatives of Γ, Δ . This is a set of parallel process contexts which represent the closures generated by the inductive case of typed context bisimilarity. These parallel representatives will be useful to simplify proofs for \approx .

Definition 6.5 (Parallel representatives). Let Γ and Δ be typing environments defined as $\Gamma = \{u_i:B_i\}_{i \in I}$ and $\Delta = \{x_j:A_j\}_{j \in J}$, respectively. We say that K is a *parallel representative* in $\mathcal{K}_{\Gamma;\Delta \vdash T}$ if

$$K \equiv (\mathbf{v}\tilde{u}, \tilde{x}) \left(\bullet \mid \prod_{i \in I} !u_i(y_i).R_i \mid \prod_{j \in J} S_j \right)$$

with $\vdash R_i :: y_i:B_i$ and $\vdash S_j :: x_j:A_j$, for every $i \in I$ and $j \in J$.

Clearly, for every left-hand side typing there may be many parallel representatives, corresponding to different implementations of the required behaviors. It is easy to see that parallel representatives are well-typed: if $K \in \mathcal{K}_{\Gamma;\Delta \vdash T}$ then $\bullet_{\Gamma;\Delta \vdash T}; \cdot \vdash K :: T$. In fact, filling in a context $K \in \mathcal{K}_{\Gamma;\Delta \vdash T}$ with a process $\Gamma, \Delta \vdash P :: T$ will lead to process $\vdash K[P] :: T$, which requires nothing from its environment. This is the essence of the desired simplification, formalized by the following proposition. It allows us to convert an (inductive) proof under non-empty typing environments Γ, Δ into a (coinductive) proof under empty environments, with processes enclosed within parallel contexts.

Proposition 6.3. $\Gamma; \Delta \vdash P \approx Q :: T$ implies $\vdash K[P] \approx K[Q] :: T$, where K is any parallel representative in $\mathcal{K}_{\Gamma;\Delta \vdash T}$, as in [Definition 6.5](#).

Proof. See [Appendix B.1](#) (Page 292) for details. \square

Based on the logical interpretation, we introduce a notion of “continuation relation” for pairs of typed processes. This will be useful to define and reason about type-respecting relations. Below, $\mathcal{I}_{\Gamma;\Delta \vdash T}$ stands for the relation

$$\{(P, Q) : \Gamma; \Delta \vdash P, Q :: T\}$$

which collects pairs of processes with identical left- and right-hand side typings.

Definition 6.6. Using \boxtimes to range over \otimes, \multimap and \boxplus to range over $\oplus, \&$, we define the type-respecting relation $\mathcal{W}_{\vdash x:A}$ by induction on the right-hand side typing, as follows:

$$\begin{aligned} \mathcal{W}_{\vdash x:1} &= \mathcal{I}_{\vdash x:1} & \mathcal{W}_{\vdash x:A \boxtimes B} &= \mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B} \\ \mathcal{W}_{\vdash x:!A} &= \mathcal{I}_{\vdash x:!A} & \mathcal{W}_{\vdash x:A \boxplus B} &= \mathcal{I}_{\vdash x:A} \cup \mathcal{W}_{\vdash x:A} \cup \mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B}. \end{aligned}$$

This way, e.g., the continuation relation for $x:A \otimes B$ is $\mathcal{I}_{\vdash x:B} \cup \mathcal{W}_{\vdash x:B}$: it contains all pairs typed by $\vdash x:B$ (as processes of type $x:A \otimes B$ are to be typed by $x:B$ after the output action) as well as those pairs in the continuation relation for $x:B$.

We now prove that \approx is a contextual relation. That is, \approx is a congruence with respect to the typed contexts associated to [Definition 6.3](#).

Lemma 6.1 (Contextuality of \approx). *Typed context bisimilarity is a contextual relation, in the sense of [Definition 6.3](#).*

Proof. The proof proceeds by coinduction, showing a typed context bisimulation for each of the conditions associated to [Definition 6.3](#). We shall exploit the proof technique given by [Proposition 6.3](#), which allows to consider \approx under empty left-hand side contexts, for pairs of processes enclosed within appropriate parallel representatives. As a result, it suffices to consider only some of the conditions in [Figure 5](#); see [Remark 6.1](#). Most cases are easy; below we detail one of them: closure with respect to output, Item (2). (See [Appendix B.2](#), Page 293 for other cases.)

We have to show that $\Gamma; \Delta \vdash P \approx Q :: y:A$ implies

$$\Gamma; \Delta, \Delta' \vdash \bar{x}(y).(P \mid S) \approx \bar{x}(y).(Q \mid S) :: x:A \otimes B$$

for any S, x, B, Δ' such that $\Gamma; \Delta' \vdash S :: x:B$. Using [Proposition 6.3](#), this can be simplified, and it suffices to show that $\vdash K_1[P] \approx K_1[Q] :: y:A$ implies

$$\vdash K_2[\bar{x}(y).(K_1[P] \mid S)] \approx K_2[\bar{x}(y).(K_1[Q] \mid S)] :: x:A \otimes B$$

where $K_1 \in \mathcal{K}_{\Gamma;\Delta \vdash y:A}$ and $K_2 \in \mathcal{K}_{\cdot;\Delta' \vdash x:A \otimes B}$.

Let $M = K_2[\bar{x}(y).(K_1[P] \mid S)]$ and $N = K_2[\bar{x}(y).(K_1[Q] \mid S)]$. Define

$$\mathcal{R}_2 = \{(M, N) : \vdash K_1[P] \approx K_1[Q] :: y:A, K_1 \in \mathcal{K}_{\Gamma;\Delta \vdash y:A}, K_2 \in \mathcal{K}_{\cdot;\Delta' \vdash x:A \otimes B}\} \cup \mathcal{W}_{\vdash x:B}$$

We show that \mathcal{R}_2 is a typed context bisimulation.

Suppose M moves first: $M \xrightarrow{\alpha} M'$. We must find a matching action from N such that $N \xrightarrow{\alpha} N'$. There are two possibilities for α : either $\alpha = \tau$ or $\alpha = \bar{x}(y)$. In the first case, we have $M \xrightarrow{\tau} K_3[\bar{x}(y).(K_1[P] \mid S)] = M'$, where $K_2 \xrightarrow{\tau} K_3$. Since K_2

occurs identically in N by construction, this action can be matched and we have $N \Longrightarrow K_4[\bar{x}(y).(K_1[Q] \mid S)] = N'$, where $K_2 \Longrightarrow K_4$. Subject reduction (Theorem 3.1) ensures both $K_3 \in \mathcal{K}_{\Gamma; \Delta \vdash y:A}$ and $K_4 \in \mathcal{K}_{\cdot; \Delta' \vdash x:A \otimes B}$, and so $(M', N') \in \mathcal{R}_2$.

In the second case we $M \xrightarrow{\bar{x}(y)} K_2[K_1[P] \mid S] = M'$. Process N can match this action, followed by zero or more reductions: $N \xrightarrow{\bar{x}(y)} K_4[K_3[Q'] \mid S'] = N'$, where $K_2 \Longrightarrow K_4$, $K_1 \Longrightarrow K_3$, $Q \Longrightarrow Q'$, and $S \Longrightarrow S'$. (Recall that K_1 and K_2 are parallel contexts, and so they are able to interact.) Theorem 5.1 ensures that these reductions are finite. Since $\vdash K_1[P] \approx K_1[Q] :: y:A$, and because of τ -closedness, we have $\vdash K_1[P] \approx K_3[Q'] :: y:A$. Subject reduction (Theorem 3.1) ensures $\vdash S, S' :: x:B$. Following the output clause of \approx , we consider the closure of M' and N' with a process L such that $y:A \vdash L :: -:1$. Such closures correspond to $K_2[(\nu y)(K_1[P] \mid L) \mid S]$ and $K_4[(\nu y)(K_3[Q'] \mid L) \mid S']$, respectively. We verify that the type of these closures is indeed $x:B$, as required by the output clause. Since $\vdash K_1[P], K_3[Q'] :: y:A$, these processes can be composed with L , and we obtain

$$\vdash (\nu y)(K_1[P] \mid L), (\nu y)(K_3[Q'] \mid L) :: -:1$$

The desired pair of processes can be obtained via an independent parallel composition with S, K_2, S' , and K_4 , respectively:

$$\vdash K_2[(\nu y)(K_1[P] \mid L) \mid S], K_4[(\nu y)(K_3[Q'] \mid L) \mid S'] :: x:B$$

Hence, $(K_2[(\nu y)(K_1[P] \mid L) \mid S], K_4[(\nu y)(K_3[Q'] \mid L) \mid S']) \in \mathcal{R}_2$ and we are done. The reasoning when N moves first is completely symmetric. \square

We now state τ -inertness, a property of transition systems which follows as a direct consequence of the results of our framework, in particular, confluence (Theorem 5.2) and the definition of typed context bisimilarity. Following Groote and Sellink [20], this property may be stated in a general way:

Definition 6.7 (τ -Inertness). Let $(\mathcal{P}, \longrightarrow)$ be a transition system, where \mathcal{P} is a set of states and $\longrightarrow \subseteq \mathcal{P} \times \mathcal{P}$. Also, let \sim stand for an equivalence relation on the elements of \mathcal{P} . We say that $(\mathcal{P}, \longrightarrow)$ is τ -inert with respect to \sim if $P \longrightarrow P'$ implies $P \sim P'$.

τ -Inertness is typically defined for labeled transition systems with a designated internal action τ , hence its name. In our case, since the LTS and the reduction relation coincide, we can safely work with reductions, and show that the class of well-typed processes is τ -inert with respect to \approx . Intuitively, τ -inertness says that reduction does not change the behavior of a process. It is therefore a property relevant for verification, as it ensures that well-typed processes can perform arbitrarily many reductions remaining in the same equivalence class; this is strengthened by the fact that termination (Theorem 5.1) ensures that these reductions are only finitely many. Adapting Definition 6.7 to our setting, we have:

Lemma 6.2 (τ -Inertness wrt \approx). Let P be a process such that $\Gamma; \Delta \vdash P :: T$. Suppose $P \longrightarrow P'$. Then $\Gamma; \Delta \vdash P \approx P' :: T$.

Proof. By coinduction, exhibiting an appropriate typed context bisimulation. Using Proposition 6.3, we work under an empty left-hand side typing. We thus define a type-respecting relation containing $(K[P], K[P'])$, for any $K \in \mathcal{K}_{\Gamma; \Delta \vdash T}$ (letting Id to stand for the identity relation):

$$\mathcal{R} = \{(K[P], K[P']) : P \longrightarrow P', K \in \mathcal{K}_{\Gamma; \Delta \vdash T}\} \cup Id \cup \mathcal{W}_{\vdash T}$$

Notice that by assumption, $\vdash K[P] :: T$; by subject reduction (Theorem 3.1) $\vdash K[P'] :: T$. We show that \mathcal{R} is a typed context bisimilarity. Suppose $K[P]$ moves first, i.e., $K[P] \xrightarrow{\alpha} M$, for some α, M . We must show a matching action $K[P'] \xrightarrow{\alpha} N$. We distinguish two cases, when $\alpha \neq \tau$ and when $\alpha = \tau$:

- If $\alpha \neq \tau$ then, necessarily, the action is related to the type in type assignment T . Appropriate inversion lemmas (Lemma 3.2) can be used to determine the actual label of α . Now, we know that $\vdash K[P], K[P'] :: T$ and that the only difference between $K[P]$ and $K[P']$ is an internal action; since $\alpha \neq \tau$, these conditions ensure that $K[P']$ can match the action α and that there exists an N such that $K[P'] \xrightarrow{\alpha} K'[P']$, where $K \Longrightarrow K'$. The analysis concludes by a case analysis on the shape of T ; depending of T , the definition of \approx determines the actual shape of the derivatives that should be found in \mathcal{R} . All cases are easy (output, input, and replicated input require suitable process closures) and covered by the definition of $\mathcal{W}_{\vdash T}$, which ensures that $(M, N) \in \mathcal{W}_{\vdash T}$.
- If $\alpha = \tau$ then there are two subcases: $M \equiv K[P']$ (i.e., α is the same τ action that leads from $K[P]$ to $K[P']$) and $M \not\equiv K[P']$ (i.e., α corresponds to a different τ action from $K[P]$). In the first subcase, $K[P']$ can trivially match this reduction with zero reductions, i.e., $K[P'] \Longrightarrow K[P'] = N$. Since the pair $(K[P'], K[P'])$ is in \mathcal{R} we are done. In the second subcase, $K[P']$ is able to match this τ action because of confluence (Theorem 5.2). Call τ_1 the τ action from $K[P]$ to $K[P']$, and let α be τ_2 . That is, $K[P]$ can exercise both τ_1 and τ_2 . Confluence ensures that if $K[P]$ performs τ_1 first, then its derivative $K[P']$ can still exercise τ_2 —this internal action is not discarded. Therefore, if $K[P]$ challenges $K[P']$ with τ_2 , confluence ensures that $K[P']$ can perform τ_2 , possibly preceded and followed by other internal actions.

A matching action $K[P'] \Longrightarrow N$, in which the weak transition contains τ_2 , thus exists, and it is easy to see that $(M, N) \in \mathcal{R}$, and we are done.

Now suppose that $K[P']$ moves first, i.e., that $K[P'] \xrightarrow{\alpha} N$. We must show a matching action $K[P] \xrightarrow{\alpha} M$. Since $K[P']$ is a τ -derivative of $K[P]$, it is easy to show that $K[P]$ can always match any action from $K[P']$: $K[P] \longrightarrow K[P'] \xrightarrow{\alpha} N$, for any α, N . This can be rewritten as $K[P] \xrightarrow{\alpha} N$ and we are done. \square

7. Applications

In this section, we first establish the soundness of proof conversions with respect to typed context bisimilarity, and then introduce a behavioral characterization of type isomorphisms. Besides clarifying further the intrinsic properties of the logical interpretation of session types, these applications illustrate the interplay of typed context bisimilarity and the properties of the type system (subject reduction, progress, termination, confluence).

7.1. Soundness of proof conversions

Recall that, by Definition 4.1, \simeq_c stands for the congruence on typed processes induced by *proof conversions*. We now show *soundness* of \simeq_c with respect to \approx , that is, we show that processes extracted from proof conversions are typed contextually bisimilar.

Before formally stating and proving this claim, we provide some intuitions on it. Consider the process equality (15) in Fig. 4 (Page 261). It corresponds to the interplay of rules (Tcut) and (T \oplus L), under typing assumptions $\Gamma; \Delta_1 \vdash \hat{D} :: x:C$, $\Gamma; \Delta_2, y:A, x:C \vdash \hat{E} :: T$, and $\Gamma; \Delta_2, y:A, x:C \vdash \hat{F} :: T$. Letting $\Delta = \Delta_1, \Delta_2$, we have:

$$\Gamma; \Delta, y:A \oplus B \vdash \underbrace{(\nu x)(\hat{D} \mid y.\text{case}(\hat{E}, \hat{F}))}_{(1)} \simeq_c \underbrace{y.\text{case}((\nu x)(\hat{D} \mid \hat{E}), (\nu x)(\hat{D} \mid \hat{F}))}_{(2)} :: T$$

with linear environments Δ_1, Δ_2 , and non-linear environment Γ , and types T, A, B, C .

Read from (1) to (2), this conversion can be interpreted as the “promotion” of the choice at y , which causes \hat{D} to get “delayed” as a result. However, such a delay is seen to be only apparent once we examine the individual typing of \hat{D} and the whole typing derivation. The first typing assumption says that \hat{D} is able to offer behavior C at x (a free name in \hat{D}), as long as it is placed in a context in which the behaviors described by names in Γ, Δ_1 are available. The left-hand side typing for both (1) and (2) says that they can offer some behavior T , as long as the behaviors declared in Γ, Δ and session $A \oplus B$ at y are provided. Crucially, since x is private to (1), type assignment T cannot correspond to $x:C$. That is, even if \hat{D} is at the top-level in (1) its behavior on x may not be immediately available. Also because of the left-hand side typing, we know that (1) and (2) are only able to interact with some selection at y ; only then, \hat{D} will be able to interact with either \hat{E} or \hat{F} , whose behavior depends on the presence of behavior C at x . A conversion of (1) into (2) could be seen as a “behavioral optimization” if one considers that (2) has only one available prefix, while (1) has two parallel components.

For all proof conversions, the apparent phenomenon of “prefix promotion” induced by proof conversions can be explained along the above lines. In our soundness result (Theorem 7.1 below), the crucial point is capturing the fact that some top-level processes may not be able to *immediately* exercise their behavior (cf. \hat{D} in (1) above). Recall that $\mathcal{I}_{\Gamma; \Delta \vdash T}$ stands for the relation which collects pairs of processes with identical left- and right-hand side typings. Also, we use the continuation relations $\mathcal{W}_{\vdash x:A}$ (cf. Definition 6.6).

Theorem 7.1 (Soundness of proof conversions). *Let P, Q be processes such that*

- (i) $\Gamma; \Delta \vdash D \rightsquigarrow P :: T$;
- (ii) $\Gamma; \Delta \vdash E \rightsquigarrow Q :: T$;
- (iii) $P \simeq_c Q$. Then, $\Gamma; \Delta \vdash P \approx Q :: T$.

Proof. By coinduction, exhibiting appropriate typed context bisimulations for each proof conversion. In the bisimulation game, we exploit termination of well-typed processes (Theorem 5.1) to ensure that actions can be matched with finite weak transitions, and subject reduction (Theorem 3.1) to ensure type preservation under reductions.

We detail the case for the first proof conversion in Fig. D.6—see Appendix C.1 (Page 295) for other cases. This proof conversion corresponds to the interplay of rules (T \otimes R) and (Tcut). We have to show that $\Gamma; \Delta \vdash M \approx N :: z:A \otimes B$ where

$$\begin{aligned} \Delta &= \Delta_1, \Delta_2, \Delta_3 \quad \Gamma; \Delta_1 \vdash \hat{D} :: x:C \quad \Gamma; \Delta_2, x:C \vdash \hat{E} :: y:A \quad \Gamma; \Delta_3 \vdash \hat{F} :: z:B \\ M &= (\nu x)(\hat{D} \mid \bar{z}(y).(\hat{E} \mid \hat{F})) \quad N = \bar{z}(y).((\nu x)(\hat{D} \mid \hat{E}) \mid \hat{F}) \end{aligned} \tag{25}$$

Using Proposition 6.3, we have to show that for every $K \in \mathcal{K}_{\Gamma; \Delta}$, we have $\vdash K[M] \approx K[N] :: z:A \otimes B$. In turn, this implies exhibiting a typed context bisimulation \mathcal{R} containing the pair $(K[M], K[N])$. We define $\mathcal{R} = \mathcal{W}_{\vdash z:A \otimes B} \cup S \cup S^{-1}$, with

$$\mathcal{S} = \{(K[M'], K[N]) : M \Longrightarrow M', K \in \mathcal{K}_{\Gamma; \Delta}\}$$

and $\mathcal{W}_{\vdash z:A \otimes B}$ is as in Definition 6.6. Notice that \mathcal{S} is a type-respecting relation indexed by $\vdash z:A \otimes B$. In fact, using the typings in (25)—with $\Gamma = \Delta = \emptyset$ —and exploiting subject reduction (Theorem 3.1), it can be checked that for all $(P, Q) \in \mathcal{S}$ both $\vdash P :: z:A \otimes B$ and $\vdash Q :: z:A \otimes B$ can be derived.

We now show that \mathcal{R} is a typed context bisimulation. Pick any $K \in \mathcal{K}_{\Gamma; \Delta}$. Using Definition 6.5, we can assume $K = (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid \cdot)$ where

- $K_{\Gamma} \equiv \prod_{i \in I} !u_i(y_i).R_i$, with $\vdash R_i :: y_i:D_i$, for every $u_i:D_i \in \Gamma$;
- $K_{\Delta} \equiv \prod_{j \in J} S_j$, with $\vdash S_j :: x_j:C_j$, for every $x_j:C_j \in \Delta$.

Clearly, $(K[M], K[N]) \in \mathcal{S}$, and so it is in \mathcal{R} . Now, suppose $K[M]$ moves first: $K[M] \xrightarrow{\alpha} M_1^*$. We have to find a matching action α from $K[N]$, i.e., $K[N] \xrightarrow{\alpha} N_1^*$. Since $\vdash K[M] :: z:A \otimes B$, we have two possible cases for α :

- 1 Case $\alpha = \tau$. We consider the possibilities for the origin of the reduction:
 - (a) $K_{\Gamma} \xrightarrow{\tau} K'_{\Gamma}$ and $K[M] \xrightarrow{\tau} K'[M]$. However, this cannot be the case, as by construction K_{Γ} corresponds to the parallel composition of input-guarded replicated processes which cannot evolve on their own.
 - (b) $K_{\Delta} \xrightarrow{\tau} K'_{\Delta}$ and $K[M] \xrightarrow{\tau} K'[M]$. Then, for some $l \in J$, $S_l \xrightarrow{\tau} S'_l$:

$$K[M] \xrightarrow{\tau} (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K'_{\Delta} \mid M) = K'[M] = M_1^*$$

Now, context K is the same in $K[N]$. Then K_{Δ} occurs identically in $K[N]$, and this reduction can be matched by a finite weak transition (Theorem 5.1):

$$K[N] \Longrightarrow (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K''_{\Delta} \mid N) = K''[N] = N_1^*$$

By subject reduction (Theorem 3.1), $\vdash S'_l :: x_l:C_l$; hence, K', K'' are in $\mathcal{K}_{\Gamma; \Delta}$. Hence, the pair $(K'[M], K''[N])$ is in \mathcal{S} (as $M \Longrightarrow M$) and so it is in \mathcal{R} .

- (c) $M \xrightarrow{\tau} M'$ and $K[M] \xrightarrow{\tau} K[M']$. Since $M = (\nu x)(\hat{D} \mid \bar{z}(y).(\hat{E} \mid \hat{F}))$, the only possibility is that there is a \hat{D}_1 such that $\hat{D} \xrightarrow{\tau} \hat{D}_1$ and $M' = (\nu x)(\hat{D}_1 \mid \bar{z}(y).(\hat{E} \mid \hat{F}))$. This way,

$$K[M] \xrightarrow{\tau} (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid M') = K[M'] = M_1^*$$

We observe that $K[N]$ cannot match this action, but $K[N] \Longrightarrow K[N]$ is a valid weak transition. Hence, $N_1^* = K[N]$. By subject reduction (Theorem 3.1), we infer that $\vdash K[M'] :: z:A \otimes B$. We use this fact to observe that the pair $(K[M'], K[N])$ is included in \mathcal{S} . Hence, it is in \mathcal{R} .

- (d) There is an interaction between M and K_{Γ} or between M and K_{Δ} : this is only possible by the interaction of \hat{D} with K_{Γ} or K_{Δ} on names in \tilde{u}, \tilde{x} . Again, the only possible weak transition from $K[N]$ matching this reduction is $K[N] \Longrightarrow K[N]$, and the analysis proceeds as in the previous case.
- 2 Case $\alpha \neq \tau$. Then the only possibility, starting from $K[M]$, is an output action of the form $\alpha = \bar{z}(y)$. This action can only originate in M :

$$K[M] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K_{\Gamma} \mid K_{\Delta} \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid \hat{F}))) = M_1^*$$

Process $K[N]$ can match this action via the following finite weak transition:

$$K[N] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K'_{\Gamma} \mid K'_{\Delta} \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}') \mid \hat{F}')) = N_1^*$$

Observe how N_1^* reflects the changes in $K[N]$ due to the possible reductions before and after the output action. By definition of \approx (output case), we consider the composition of M_1^* and N_1^* with any V such that $y:A \vdash V :: \neg:1$. Using the typings in (25) and subject reduction (Theorem 3.1), we infer both

$$\vdash M_2^* = (\nu \tilde{x}, \tilde{u})(K_{\Gamma} \mid K_{\Delta} \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid V \mid \hat{F}))) :: z:B$$

$$\vdash N_2^* = (\nu \tilde{x}, \tilde{u})(K'_{\Gamma} \mid K'_{\Delta} \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}' \mid V) \mid \hat{F}')) :: z:B$$

Hence, the pair (M_2^*, N_2^*) is in $\mathcal{W}_{\vdash z:A \otimes B}$ and so it is in \mathcal{R} .

Now suppose that $K[N]$ moves first: $K[N] \xrightarrow{\alpha} N_1^*$. We have to find a matching action α from $K[M]$: $K[M] \xrightarrow{\alpha} M_1^*$. Similarly as before, there are two cases: either $\alpha = \tau$ or $\alpha = \bar{z}(y)$. The former is as detailed before; the only difference is that reductions from $K[N]$ can only be originated in K_{Δ} ; these are matched by $K[M]$ with finite weak transitions originating in both K and in M . We thus obtain pairs of processes in \mathcal{S}^{-1} . The analysis for the case for output mirrors the given above and is omitted. \square

7.2. A behavioral characterization of session type isomorphisms

In type theory, types A and B are called *isomorphic* if there are morphisms π_A of $B \vdash A$ and π_B of $A \vdash B$ which compose to the identity in both ways—see, e.g., [18]. For instance, in the λ -calculus the types $A \times B$ and $B \times A$ are isomorphic since we can construct terms $M = \lambda x:A \times B. \langle \pi_2 x, \pi_1 x \rangle$ and $N = \lambda x:B \times A. \langle \pi_2 x, \pi_1 x \rangle$, respectively of types $A \times B \rightarrow B \times A$ and $B \times A \rightarrow A \times B$, such that both compositions $\lambda x:B \times A. (M(Nx))$ and $\lambda x:A \times B. (N(Mx))$ are equivalent (up to η -conversion) to the identity $\lambda x:B \times A. x$ and $\lambda x:A \times B. x$.

We adapt this notion to our setting, by using proofs as morphisms, and by using typed context bisimilarity to account for *isomorphisms* in linear logic.

Given a sequence of names $\tilde{x} = x_1, \dots, x_n$, below we write $P^{(\tilde{x})}$ to denote a process such that $fn(P) = \{x_1, \dots, x_n\}$.

Definition 7.1 (Type isomorphism). Two (session) types A and B are called *isomorphic*, noted $A \simeq B$, if, for any names x, y, z , there exist processes $P^{(x,y)}$ and $Q^{(y,x)}$ such that:

- (i) $\cdot; x:A \vdash P^{(x,y)} \vdash y:B$;
- (ii) $\cdot; y:B \vdash Q^{(y,x)} \vdash x:A$;
- (iii) $\cdot; x:A \vdash (\nu y)(P^{(x,y)} \mid Q^{(y,z)}) \approx [x \leftrightarrow z] \vdash z:A$; and
- (iv) $\cdot; y:B \vdash (\nu x)(Q^{(y,x)} \mid P^{(x,z)}) \approx [y \leftrightarrow z] \vdash z:B$.

Thus, intuitively, if A, B are service specifications then by establishing $A \simeq B$ one can claim that having A is as good as having B , because we can build one from the other using an isomorphism. Isomorphisms in linear logic can then be used to simplify/transform service interfaces in the π -calculus. They can also help validating our interpretation with respect to basic linear logic principles. As an example, let us consider multiplicative conjunction \otimes . A basic linear logic principle is $A \otimes B \vdash B \otimes A$. Our interpretation of $A \otimes B$ may appear asymmetric as, in general, a channel of type $A \otimes B$ is not typable by $B \otimes A$. [Theorem 7.2](#) below states the symmetric nature of \otimes as a type isomorphism: symmetry is realized by a process which *coerces* any session of type $A \otimes B$ to a session of type $B \otimes A$.

Theorem 7.2. Let A, B , and C be any type, as in [Definition 3.1](#). Then the following hold:

- (i) $A \otimes B \simeq B \otimes A$
- (ii) $(A \oplus B) \multimap C \simeq (A \multimap C) \wp (B \multimap C)$
- (iii) $!(A \wp B) \simeq !A \otimes !B$.

Proof. We give details for the proof of (i) above; see [Appendix C.2](#), [Page 298](#), for further details.

We check conditions (i)–(iv) of [Definition 7.1](#) for processes $P^{(x,y)}, Q^{(y,x)}$ defined as

$$P^{(x,y)} = x(u). \bar{y}(n). ([x \leftrightarrow n] \mid [u \leftrightarrow y])$$

$$Q^{(y,x)} = y(w). \bar{x}(m). ([y \leftrightarrow m] \mid [w \leftrightarrow x])$$

Checking (i)–(ii), i.e., $\cdot; x:A \otimes B \vdash P^{(x,y)} \vdash y:B \otimes A$ and $\cdot; y:B \otimes A \vdash Q^{(y,x)} \vdash x:A \otimes B$ is easy; rule (Tid) ensures that both typings hold for any A, B . We sketch only the proof of (iii); the proof of (iv) is analogous. Let $M = (\nu y)(P^{(x,y)} \mid Q^{(y,z)})$ and $N = [x \leftrightarrow z]$; we need to show $\cdot; x:A \otimes B \vdash M \approx N \vdash z:A \otimes B$. By [Proposition 6.3](#), we have to show that for every $K \in \mathcal{K}_{\cdot; x:A \otimes B}$, we have $\vdash K[M] \approx K[N] \vdash z:A \otimes B$. In turn, this implies exhibiting a typed context bisimulation \mathcal{R} containing $(K[M], K[N])$. Letting $\mathcal{S} = \{(R_1, R_2) : K[M] \Longrightarrow R_1, K[N] \Longrightarrow R_2\}$, we set $\mathcal{R} = \mathcal{W}_{\vdash z:A \otimes B} \cup \mathcal{S} \cup \mathcal{S}^{-1}$. Following expected lines, \mathcal{R} can be shown to be a typed context bisimulation. \square

8. Related work

Logical relations in concurrency In a concurrent/process calculi setting, logical relations (or closely related techniques) have been investigated by Berger, Honda, and Yoshida [47,3,4], Sangiorgi [41], Caires [8], and Boudol [7]. None of these works considers session types, and so the logical relations proposed in such works are very different from ours. Boudol [7] relies on the classical realizability technique (together with a type and effect system) to establish termination in a higher-order imperative language. Caires [8] proposes a semantic approach to proving soundness for type systems for concurrency, by relying on a spatial logic interpretation of types. More related to our developments are works by Yoshida, Berger, Honda [47] and by Sangiorgi [41], which aim at identifying terminating fragments of the π -calculus by using types, relying on arguments based on logical relations. The logical relations framework developed in [47] is extended in [3,4] to the case of a second-order, polymorphic π -calculus. A main result in [3,4] is a proof of termination using the method of reducibility candidates; while [3] reports a relational parametricity result, [4] puts forward a behavioral theory based on generic transitions and a fully abstract embedding of System F. All of these works consider typing disciplines different from session types; consequently, associated semantic interpretations of types are very different from ours, and rely on constraints on the syntax

and the types of processes. In sharp contrast to [47,41], which aim at type disciplines that guarantee termination, here we started from a well-established type discipline for the π -calculus and have used linear logical relations to show termination and confluence of well-typed processes. We have shown how the interpretation of intuitionistic linear logic as session types in [11] leads to intuitive logical relations, naturally defined on the structure of types. In this sense, our approach is more principled than in [47,41], as it is not an adaptation of the method, but rather an instantiation of the method on our canonical linear type structure.

Logical interpretations of session types Dal Lago and Di Giambardino [14] introduce an interpretation of session types as soft linear logic propositions [28]. As a result, the exponential “!” is treated following a non-canonical discipline that uses two different typing environments. Hence, typing rules and judgments in [14] are rather different from ours. A bound on the length of reductions starting from well-typed-processes is obtained; the proof uses techniques from Implicit Computational Complexity. Neither confluence, observational equivalences, nor issues of inference permutability and type isomorphisms are addressed in [14]. Although here we do not provide a similar bound, it is remarkable that our proof of termination follows *only* the principles and properties of [11]; in contrast to [14], our proof appeals to well-known technical devices, and allows us to retain a standard, intuitive treatment of “!”. This is particularly desirable for extensions/generalizations of our logical interpretation of session types, such as the proposed in [45,35].

Loosely related is Mazurak and Zdanciewicz’s Lollipop [29], a functional language with support for concurrency based on control operators. Lollipop’s operational semantics is based on a runtime process calculus; thread communication is defined in terms of protocol types which are given a *classic* linear logic interpretation. As in our case, type soundness, strong normalization, and confluence results hold for Lollipop; however, the details of the associated proof techniques are rather different from ours.

Determinacy and confluence in process calculi In term rewriting systems such as the λ -calculus, determinacy and confluence are well-understood issues, and typically rely on (unlabeled) reduction semantics. For process calculi, a semantics given in terms of labelled transition systems is often useful, for it describes the interaction of processes with their environment. As a result, notions of determinacy and confluence for process calculi typically account for those labels, thus setting a major difference with respect to traditional notions. It is worth noticing that our notion of confluence (Definition 5.2) considers only weak transitions based on internal behavior, and so it is closer to classical definitions of confluence rather than to the definitions used in process calculi. Early studies of determinacy and confluence for process calculi are due to Milner, in the setting of CCS [30]; his interest was on proper definitions of such notions, focusing on syntactic conditions on process constructs so as to build determinate, confluent systems by construction. There is a close relationship between determinacy, confluence, τ -inertness and the given notion of equivalence; Groote and Sellink [20] provide a general study on such a relationship, focusing on the impact of such notions on process verification. Milner’s approach to confluence was extended to the π -calculus by Walker and Philippou [36], and by Nestmann [32] who characterizes (forms of) confluence in terms of so-called *port uniqueness* for polarized name-passing, which is ensured by static typing. Most related to our work is the work by Kouzapas et al. [27], which adapts Walker and Philippou’s techniques to establish session determinacy and confluence for a session-typed *asynchronous* π -calculus. The above mentioned differences in the definition of determinacy and confluence prevent detailed comparisons with our confluence result, which relies on reductions and is shown using logical relations.

Typed behavioral equivalences Previous works on behavioral equivalences for *typed* process calculi have considered a number of different typing disciplines. For instance, behavioral theories for calculi with linear types (e.g., [26]), input/output types (e.g., [6,37,16]), subtyping with name matching (e.g., [21]), and polymorphic types (e.g., [38]) have been put forward. Still, the only work on behavioral equivalences for binary session-typed processes we are aware of is [27]. It studies the behavioral theory of a π -calculus with *asynchronous*, *event-based* binary session communication. The aim is to capture the distinction between order-preserving communications (those inside already established connections) and non-order-preserving communications (those outside such connections). The behavioral theory in [27] accounts for principles for prefix commutation that appear similar to those induced by our proof conversions. However, the origin and nature of these commutations are quite different. In fact, in [27] prefix commutation arises from the above-mentioned distinction, whereas commutations in our (synchronous) framework are due to causality relations captured by types. Loosely related to typed context bisimilarity is [48], where a form of *linear bisimilarity* is proposed; following a linear type structure, it treats some visible actions as internal actions, thus leading to an equivalence larger than standard bisimilarity which is a congruence.

9. Concluding remarks

In this paper, we have introduced a theory of linear logical relations and a notion of typed behavioral equivalences for session-typed, concurrent processes. These developments extend the interpretation of linear logic propositions as session types developed by Caires and Pfenning in [11].

Our theory of linear logical relations is remarkably similar to that for functional languages; although in our setting session types are assigned to names (and not to terms), our linear logical relations are defined on the structure of types, relying both on process reductions and labeled transitions. A main application of this theory is a proof that well-typed processes are both strongly normalizing (Theorem 5.1) and confluent (Theorem 5.2). In practice, certifying termination and confluence of

session-typed programs is important. We believe the extended correctness guarantees given by our results could be highly beneficial for the increasingly growing number of practical implementations (libraries, programming language extensions) based on session types foundations—see, e.g., [25,33,39].

We have also presented a behavioral theory for session-typed processes. We introduced *typed context bisimilarity*, a novel labeled bisimilarity over typed processes, and studied its properties. Our definition follows from the intuitive meaning of type judgments, and is stated in the style of conventional definitions for untyped processes. In addition to studying its main properties, we have illustrated this typed observational equivalence in two applications, which strengthen the properties of the logic interpretation established in [11]. On the one hand, we have shown soundness of *proof conversions* with respect to observational equivalence—an issue left open in [11] (Theorem 7.1). On the other hand, we studied *type isomorphisms* resulting from linear logic equivalences in our setting (Theorem 7.2). The basic properties of the interpretation—especially, the combination of subject reduction and termination—were of the essence in the proofs of both applications.

There are some intuitive similarities in the definitions used in formalizing our theory of linear logical relations and those required for developing our behavioral theory. We have given a formal connection between the two topics in [9,10], where the linear logic relations developed here are generalized to the case of *parametric polymorphism*. In this extended setting, existential and universal quantification over types are interpreted as a form of session type-passing; using logical relations we have characterized barbed congruence in a sound and complete way. In future work, we plan to adapt the results here presented to the case of the interpretation of session types into *classical* linear logic, as defined in [12, §5] and [46].

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions. This research was supported by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through grants INTERFACES NGN-44/2009 (Carnegie Mellon Portugal Program), SFRH/BD/33763/2009 (Toninho), SFRH/BPD/84067/2012 (Pérez), and CITI.

Appendix A. Proofs of Section 5 (Logical Relations)

Below, we write $P \not\rightarrow$ to mean that P cannot reduce; it can perform visible actions, though. Also, we write $P \rightarrow^k P'$ to denote a reduction sequence of length k from P to P' . Given a process $P \Downarrow$, we write $\text{mlen}(P)$ to stand for the length of the longest reduction sequence originating from P . Given terminating processes P_1, \dots, P_n , notation $\text{mlen}(P_1, \dots, P_n)$ stands for $\text{mlen}(P_1) + \dots + \text{mlen}(P_n)$.

A.1. Proof of Proposition 5.7

We repeat the statement in Page 266:

Proposition Appendix A.1 (Proposition 5.7). *Let P, Q be well-typed processes. If $P \in \mathcal{L}[T]$ and $P \equiv_! Q$ then $Q \in \mathcal{L}[T]$.*

Proof. By induction on the definition of $P \equiv_! Q$ (Definition 5.3). Given Proposition 5.5, it suffices to consider only the sharpened replication axioms. In each case, we use an auxiliary induction on the structure of T , which relies on Proposition 5.2 and on the operational correspondence between P and Q given by Proposition 5.1.

- Axiom (1): Then we have two sub-cases. In the first one, we have

$$\begin{aligned} P &= (\nu u)(!u(z).P_1 \mid (\nu y)(P_2 \mid P_3)) \\ Q &= (\nu y)((\nu u)(!u(z).P_1 \mid P_2) \mid (\nu u)(!u(z).P_1 \mid P_3)) \end{aligned}$$

Hence, sub-process $!u(z).P_1$ has been distributed to the unguarded processes P_2 and P_3 . We proceed by induction on the structure of T . Each case proceeds by showing that Q satisfies termination, well-typedness, and operational correspondence requirements stated in Definition 5.4. For the latter, we use Proposition 5.1(1) and 5.1(2). We have six cases to check; we detail only some of them as the rest is similar.

Case $P \in \mathcal{L}[z:1]$. Then $P \Downarrow$ and $\cdot \vdash P :: z:1$ and for all P' such that $P \Rightarrow P'$ and $P' \not\rightarrow$ it implies that $P' \equiv_! \mathbf{0}$. First, by Proposition 5.2, we have that $Q \Downarrow$. Now, since $\cdot \vdash P :: z:1$ it is easy to show that there exists a typing derivation for $\cdot \vdash Q :: z:1$. Finally, by Proposition 5.1(1), we know that Q can match any reduction from P . Therefore, there exists a Q' such that $Q \Rightarrow Q'$ and $Q' \not\rightarrow$ and $P' \equiv_! Q'$. By transitivity of $\equiv_!$, we have that $Q' \equiv_! \mathbf{0}$ and so $Q \in \mathcal{L}[z:1]$, as desired.

Case $P \in \mathcal{L}[z:A \multimap B]$. Then $P \Downarrow$ and $\cdot \vdash P :: z:A \multimap B$. Hence, by Proposition 5.6, P has an input action on z . Moreover, by Definition 5.4, for all P', y such that $P \xrightarrow{z(y)} P'$ it implies that $\forall R \in \mathcal{L}[y:A]. (\nu y)(P' \mid R) \in \mathcal{L}[z:B]$. First, by Proposition 5.2, we have that $Q \Downarrow$. Now, since $\cdot \vdash P :: z:A \multimap B$, it is easy to show that there exists a typing derivation for $\cdot \vdash Q :: z:A \multimap B$. Finally, by Proposition 5.1(1) and 5.1(2), we know that Q can match any reduction/transition from P . Therefore, there exists a Q' such that $Q \xrightarrow{z(y)} Q'$ and $P' \equiv_! Q'$. Now, by induction hypothesis we have that $\forall R \in \mathcal{L}[y:A]. (\nu y)(Q' \mid R) \in \mathcal{L}[z:B]$, and so $Q \in \mathcal{L}[z:A \multimap B]$, as desired.

Case $P \in \mathcal{L}[z:A \otimes B]$. Then $P \Downarrow$ and $\cdot \vdash P :: z:A \otimes B$. Hence, by [Proposition 5.6](#), P has an output action on z . Moreover, by [Definition 5.4](#), for all P' , y such that $P \xrightarrow{\bar{z}(y)} P'$ it implies that there exist P_1, P_2 such that $P' \equiv P_1 \mid P_2$ and $P_1 \in \mathcal{L}[y:A]$ and $P_2 \in \mathcal{L}[z:B]$. First, by [Proposition 5.2](#), we have that $Q \Downarrow$. Now, since $\cdot \vdash P :: z:A \otimes B$, it is easy to show that there exists a typing derivation for $\cdot \vdash Q :: z:A \otimes B$. Now, by [Proposition 5.1\(1\)](#) and [5.1\(2\)](#), we know that Q can match any reduction/transition from P . Therefore, there exists a Q' such that $Q \xrightarrow{\bar{z}(y)} Q'$ and $P' \equiv Q'$. Now, by transitivity we have that $Q' \equiv P_1 \mid P_2$, and so $Q \in \mathcal{L}[z:A \otimes B]$, as desired.

Case $P \in \mathcal{L}[z:A]$. Similar to the case $P \in \mathcal{L}[z:1]$.

The second sub-case is symmetric to the first one, with P defined as Q and Q defined as P . As such, sub-process $!u(z).P_1$ has been “factorized” from the process expression. The analysis follows the lines of the first case and is omitted.

- Axiom (2): Then we have two sub-cases. In the first one, we have:

$$\begin{aligned} P &= (\nu u)(!u(y).P_1 \mid (\nu v)(!v(z).P_2 \mid P_3)) \\ Q &= (\nu v)((!v(z).(\nu u)(!u(y).P_1 \mid P_2)) \mid (\nu u)(!u(y).P_1 \mid P_3)) \end{aligned}$$

Similarly as before, sub-process $!u(y).P_1$ has been distributed to the unguarded process P_3 and to the input-guarded replicated process $!v(z).P_2$. We proceed by induction on the structure of T . Each case proceeds by showing that Q satisfies the requirements stated in [Definition 5.4](#). The analysis mirrors the one given above for Axiom (1), using [Proposition 5.2](#), observing that typability of P under some type assignment T implies typability of Q under T , and exploiting the operational correspondence between P and Q given by [Proposition 5.1\(1\)](#) and [5.1\(2\)](#).

In the second sub-case, P defined as Q and Q defined as P . As such, sub-process $!u(y).P_1$ has been factorized from the process expression. The analysis follows the lines of the first sub-case and is omitted.

- Axiom (3): Then we have two sub-cases. In the first one, we have

$$P = (\nu u)(!u(y).P_1 \mid P_2) \quad \text{with } u \notin \text{fn}(P_2) \quad Q = P_2$$

Hence, sub-process $!u(y).P_1$ is discarded, as it cannot be invoked by P_2 . We proceed by induction on the structure of T . Each case proceeds by showing that Q satisfies the requirements stated in [Definition 5.4](#). The crucial point is to observe that since $u \notin \text{fn}(P_2)$ then every reduction/transition from P originates in P_2 , and so they can be trivially matched by Q . As a consequence, P belongs to $\mathcal{L}[z:T]$, for some $z \neq u$. We have six cases to check; we detail two of them, the others are similar:

Case $P \in \mathcal{L}[z:1]$. Then $P \Downarrow$ and $\cdot \vdash P :: z:1$ and for all P' such that $P \Rightarrow P'$ and $P' \not\Rightarrow$ it implies that $P' \equiv \mathbf{0}$. First, by [Proposition 5.2](#), we have that $Q \Downarrow$. Now, since $\cdot \vdash P :: z:1$ it is possible to show that $\cdot \vdash Q :: z:1$. Notice also that since $u \notin \text{fn}(P_2)$, none of the reductions from P to P' is a synchronization on u . Hence, every reduction of P originates in P_2 , and since $Q = P_2$, the thesis trivially holds.

Case $P \in \mathcal{L}[z:A \multimap B]$. Then $P \Downarrow$ and $\cdot \vdash P :: z:A \multimap B$. Hence, by [Proposition 5.6](#), P has an input action on z . Moreover, by [Definition 5.4](#), for all P' , y such that $P \xrightarrow{z(y)} P'$ it implies that $\forall R \in \mathcal{L}[y:A].(\nu y)(P' \mid R) \in \mathcal{L}[z:B]$. First, by [Proposition 5.2](#), we have that $Q \Downarrow$. Now, since $\cdot \vdash P :: z:A \multimap B$ then it can be shown that $\cdot \vdash Q :: z:A \multimap B$. Now, since $u \notin \text{fn}(P_2)$, none of the reductions/transition from P to P' is a synchronization on u . Hence, every reduction and transition of P originates in P_2 , and since $Q = P_2$, we immediately infer that $Q \in \mathcal{L}[z:A \multimap B]$, as desired.

The second sub-case is the symmetric of the first one, with P defined as Q and Q defined as P . That is, process Q is the extension of $P = P_2$ with a process $!u(y).P_1$ that it cannot invoke. Notice that we assume well-typed processes, and so the extended process Q is well-typed as well. The analysis follows the lines of the first case and is omitted. \square

A.2. Proof of [Proposition 5.8](#)

We repeat the statement in [Page 266](#):

Proposition Appendix A.2 ([Proposition 5.8](#)). Let $\cdot \vdash P :: z:A$ be a well-typed process. If $P \rightarrow_x P_1$ and $P \rightarrow_y P_2$ and $P_1 \neq P_2$ then there exist P'_1, P'_2 such that $P_1 \rightarrow_y P'_1$ and $P_2 \rightarrow_x P'_2$.

Proof. By a case analysis on the different ways in which two different reductions on private names can arise from the process P . By assumption, $P \equiv (\nu \tilde{n})P_0$ where x and y occur in \tilde{n} and both

$$(\nu \tilde{n})P_0 \rightarrow_x (\nu \tilde{n})P'_0 = P_1 \quad (\nu \tilde{n})P_0 \rightarrow_y (\nu \tilde{n})P''_0 = P_2$$

Observe that [Theorem 3.1](#) ensures both $\cdot \vdash P_1 :: z:A$ and $\cdot \vdash P_2 :: z:A$. Both these reductions are inferred by the following reduction rule (cf. [Fig. 1](#)):

$$\frac{P \rightarrow P'}{(\nu z)P \rightarrow (\nu z)P'}$$

We must show that there exist P'_1, P'_2 such that $P_1 \rightarrow_y P'_1$ and $P_2 \rightarrow_x P'_2$. In our process model there are four main possibilities for enabling a reduction (namely, communication, a shared server invocation, a choice, and forwarding) which are realized by five reduction rules. Consequently, the required analysis involves 16 cases, resulting from the combination of these four main possibilities, using typing inversion. In all cases, we exploit [Theorem 3.1](#) (to ensure type preservation) and [Theorem 3.2](#) (which guarantees that processes are not stuck and reductions can proceed).

Case 1. *The reduction on x is a communication; the reduction on y is a choice.*

Then we would have two possibilities:

$$P \equiv (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid y.\text{inl}; P_4 \mid y.\text{case}(P_5, P_6) \mid R_z)$$

$$P \equiv (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid y.\text{inr}; P_4 \mid y.\text{case}(P_5, P_6) \mid R_z)$$

for some P_1, \dots, P_6, R_z , with x, y , and w occurring in \tilde{n} . Here and in the following, we write R_z to denote the free occurrence of name z . Let us consider only the first possibility; the second is analogous. The two enabled reductions correspond to independent sessions which are inherently non-interfering from each other:

$$P \rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid y.\text{inl}; P_4 \mid y.\text{case}(P_5, P_6) \mid R_z) = P'$$

$$P \rightarrow_y (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid P_4 \mid P_5 \mid R_z) = P''$$

Clearly, one reduction does not preclude the other:

$$P' \rightarrow_y (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid P_4 \mid P_5 \mid R_z)$$

$$P'' \rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid P_4 \mid P_5 \mid R_z)$$

and thus the thesis follows.

Case 2. *The reduction on x is a communication; the reduction on y is a forwarding.*

Then we would have:

$$P \equiv (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid [y \leftrightarrow l] \mid R_z)$$

for some P_1, \dots, P_3, R_z , with x, y , and w occurring in \tilde{n} . Also in this case, one reduction does not preclude the other. We have:

$$P \rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid [y \leftrightarrow l] \mid R_z)$$

$$\rightarrow_y (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid R_z)\{l/y\}$$

$$P \rightarrow_y (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid R_z)\{l/y\}$$

$$\rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid R_z)\{l/y\}$$

and thus the thesis follows.

Case 3. *The reduction on x is a communication; the reduction on y is a server invocation.*

Then we would have:

$$P \equiv (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid y\langle l \rangle.P_4 \mid !y(u).P_5 \mid R_z)$$

for some P_1, \dots, P_5, R_z , with x, y, w , and l occurring in \tilde{n} . Also in this case, one reduction does not preclude the other. We have:

$$P \rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid y\langle l \rangle.P_4 \mid !y(u).P_5 \mid R_z)$$

$$\rightarrow_y (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid P_4 \mid P_5\{l/u\} \mid !y(u).P_5 \mid R_z)$$

$$P \rightarrow_y (\nu \tilde{n})(x\langle w \rangle.(P_1 \mid P_2) \mid x(v).P_3 \mid P_4 \mid P_5\{l/u\} \mid !y(u).P_5 \mid R_z)$$

$$\rightarrow_x (\nu \tilde{n})(P_1 \mid P_2 \mid P_3\{w/v\} \mid P_4 \mid P_5\{l/u\} \mid !y(u).P_5 \mid R_z)$$

and thus the thesis follows.

Case 4. *The reduction on x is a communication; the reduction on y is also a communication.*

This case is similar to Case 3.

Case 5. *Both reductions, on x and y , are server invocations.*

There are two sub-cases. In the first one we would have invocations to two different servers, i.e., $x \neq y$:

$$P \equiv (\nu \tilde{n})(x\langle w \rangle.P_1 \mid !x(v).P_2 \mid y\langle l \rangle.P_3 \mid !y(u).P_4 \mid R_z)$$

for some P_1, \dots, P_4, R_z , with x, y, w , and l occurring in \tilde{n} . We have:

$$\begin{aligned}
P &\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_2\{w/v\} \mid !x(v).P_2 \mid y(l).P_3 \mid !y(u).P_4 \mid R_z) \\
&\longrightarrow_y (\mathbf{v}\tilde{n})(P_1 \mid P_2\{w/v\} \mid !x(v).P_2 \mid P_3 \mid P_4\{l/u\} \mid !y(u).P_4 \mid R_z) \\
P &\longrightarrow_y (\mathbf{v}\tilde{n})(x(w).P_1 \mid !x(v).P_2 \mid P_3 \mid P_4\{l/u\} \mid !y(u).P_4 \mid R_z) \\
&\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_2\{w/v\} \mid !x(v).P_2 \mid P_3 \mid P_4\{l/u\} \mid !y(u).P_4 \mid R_z)
\end{aligned}$$

Thus, one reduction does not preclude the other. In the second sub case, we have two invocations to the same server, i.e., $x = y$:

$$P \equiv (\mathbf{v}\tilde{n})(x(w).P_1 \mid x(l).P_2 \mid !x(v).P_3 \mid R_z)$$

We assume that x , w , and l occur in \tilde{n} . We have:

$$\begin{aligned}
P &\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_3\{w/v\} \mid x(l).P_2 \mid !x(v).P_3 \mid R_z) \\
&\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_3\{w/v\} \mid P_2 \mid P_3\{l/v\} \mid !x(v).P_3 \mid R_z) \\
P &\longrightarrow_x (\mathbf{v}\tilde{n})(x(w).P_1 \mid P_2 \mid P_3\{l/v\} \mid !x(v).P_3 \mid R_z) \\
&\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_3\{w/v\} \mid P_2 \mid P_3\{l/v\} \mid !x(v).P_3 \mid R_z)
\end{aligned}$$

Here again, one reduction does not preclude the other. Notice that the order in which the synchronizations occur is irrelevant, for the shared server is a persistent (replicated) process.

Case 6. *The reduction on x is a server invocation; the reduction on y is a communication.*

We would have

$$P \equiv (\mathbf{v}\tilde{n})(x(w).P_1 \mid !x(v).P_2 \mid y(l).(P_3 \mid P_4) \mid y(u).P_5 \mid R_z)$$

where x , y , w , and l occur in \tilde{n} . In this case, the reasoning is as in Case 3

Case 7. *The reduction on x is a server invocation; the reduction on y is a forward.*

We would have

$$P \equiv (\mathbf{v}\tilde{n})(x(w).P_1 \mid !x(v).P_2 \mid [y \leftrightarrow l] \mid R_z)$$

where x , y , and w occur in \tilde{n} . The reasoning is similar as in Case 2 above.

Case 8. *The reduction on x is a server invocation; the reduction on y is a choice.*

We would have

$$P \equiv (\mathbf{v}\tilde{n})(x(w).P_1 \mid !x(v).P_2 \mid y.\text{inl}; P_3 \mid y.\text{case}(P_4, P_5) \mid R_z)$$

where x , y , and w occur in \tilde{n} and the reasoning is similar as in Case 1 above.

Case 9. *The reduction on x is a forwarding; the reduction on y is a communication.*

This case is symmetric to Case 2 above.

Case 10. *The reduction on x is a forwarding; the reduction on y is a server invocation.*

This case is symmetric to Case 7 above.

Case 11. *Both reductions, on x and y , are forwardings.*

We would have

$$P \equiv (\mathbf{v}\tilde{n})([x \leftrightarrow w] \mid [y \leftrightarrow l] \mid R_z)$$

where x and y occur in \tilde{n} , and the thesis follows easily.

Case 12. *The reduction on x is a forwarding; the reduction on y is a choice.*

We would have two possibilities:

$$\begin{aligned}
P &\equiv (\mathbf{v}\tilde{n})([x \leftrightarrow w] \mid y.\text{inl}; P_1 \mid y.\text{case}(P_2, P_3) \mid R_z) \\
P &\equiv (\mathbf{v}\tilde{n})([x \leftrightarrow w] \mid y.\text{inr}; P_1 \mid y.\text{case}(P_2, P_3) \mid R_z)
\end{aligned}$$

for some P_1, \dots, P_3, R_z , with x and y occur in \tilde{n} . In both cases the thesis follows easily.

Case 13. *Both reductions, on x and y , are choices.*

There are four sub-cases, depending on combinations of right and left selection along x and y . We consider one particular subcase—the other three are analogous:

$$P \equiv (\mathbf{v}\tilde{n})(x.\text{inr}; P_1 \mid x.\text{case}(P_2, P_3) \mid y.\text{inl}; P_4 \mid y.\text{case}(P_5, P_6) \mid R_z)$$

for some P_1, \dots, P_6, R_z , with x , y , and w occur in \tilde{n} . The two enabled reductions correspond to independent, non-interfering sessions:

$$\begin{aligned}
P &\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_3 \mid y.\text{inl}; P_4 \mid y.\text{case}(P_5, P_6) \mid R_z) \\
&\longrightarrow_y (\mathbf{v}\tilde{n})(P_1 \mid P_3 \mid P_4 \mid P_5 \mid R_z) \\
P &\longrightarrow_y (\mathbf{v}\tilde{n})(x.\text{inr}; P_1 \mid x.\text{case}(P_2, P_3) \mid P_4 \mid P_5 \mid R_z) \\
&\longrightarrow_x (\mathbf{v}\tilde{n})(P_1 \mid P_3 \mid P_4 \mid P_5 \mid R_z)
\end{aligned}$$

Thus, one reduction does not preclude the other and the thesis follows.

Case 14. *The reduction on x is a choice; the reduction on y is a forwarding.*

This case is symmetric to Case 12 above.

Case 15. *The reduction on x is a choice; the reduction on y is a server invocation.*

This case is symmetric to Case 8 above.

Case 16. *The reduction on x is a choice; the reduction on y is a communication.*

This case is symmetric to Case 1 above. \square

A.3. Proof of Proposition 5.12

We repeat the statement in Page 267:

Proposition Appendix A.3 (Proposition 5.12). *Let P, Q be processes such that $P \in \mathcal{L}[T]$ and $Q \in \mathcal{L}[-:1]$. Then, $P \mid Q \in \mathcal{L}[T]$.*

Proof. By induction on the structure of T . First, it is worth observing that $P \in \mathcal{L}[T]$ and $Q \in \mathcal{L}[-:1]$ imply $\cdot; \cdot \vdash P :: T$ and $\cdot; \cdot \vdash Q :: -:1$, respectively. Hence, we can derive the typing $\cdot; \cdot \vdash P \mid Q :: T$ (cf. the derived rule (comp)). In fact, the type of Q indicates it cannot offer any visible action to its environment, and so it is “independent” from it.

If $T = -:1$ then $P \mid Q$ represents the parallel composition of two terminating processes that cannot interact with each other. Hence, for all R such that $P \mid Q \Longrightarrow R$ and $R \not\rightarrow$ we have that $R \equiv \mathbf{0}$, and so $P \mid Q \in \mathcal{L}[-:1]$. The cases in which $T \neq -:1$ rely on the fact that if $P \xrightarrow{\alpha} P'$ then there exists a process R such that $P \mid Q \xrightarrow{\alpha} R$. The proof is by induction on $k = \text{mlen}(Q)$. If $k = 0$ then $Q \not\rightarrow$ and for every weak transition $P \xrightarrow{\alpha} P'$, we have $P \mid Q \xrightarrow{\alpha} P' \mid Q = R$. In the inductive case, we assume $k > 0$, and so reductions (or the action α) from P may go interleaved with reductions from Q . Given $P \xrightarrow{\alpha} P'$ then by induction hypothesis there is an R' such that $P \mid Q \xrightarrow{\alpha} P' \mid Q' = R'$, with Q reducing to Q' in $k - 1$ steps. Then, if $Q' \rightarrow Q''$ we would have $P \mid Q \xrightarrow{\alpha} P' \mid Q' \rightarrow P' \mid Q''$ which is equivalent to write $P \mid Q \xrightarrow{\alpha} R$, with $R = P' \mid Q''$, and we are done. Finally, we observe that, given $P \xrightarrow{\alpha} P'$, process Q (and its derivatives) pose no difficulties when decomposing P' into smaller processes (in the case $T = z:A \otimes B$, for instance). Hence, we can conclude that if $P \in \mathcal{L}[T]$ then $P \mid Q \in \mathcal{L}[T]$, as desired. \square

A.4. Proof of Lemma 5.2

We repeat the statement in Page 267 below. In the proof, we use G, G', \dots and D, D', \dots to range over processes in \mathcal{C}_Γ and \mathcal{C}_Δ , respectively. Also, by a slight abuse of notation we write $\mathcal{L}[x:A]$ and $!z(y).\mathcal{L}[y:A]$ to denote a process included in $\mathcal{L}[x:A]$ and $\mathcal{L}[!z:A]$, respectively.

Lemma Appendix A.1 (Lemma 5.2). *If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$.*

Proof. By induction on the derivation of $\Gamma; \Delta \vdash P :: T$, with a case analysis on the last typing rule used.

Thus, we have 18 cases to check. In all of them, we use Lemma 5.1 and show that every $M = (\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D)$ with $G \in \mathcal{C}_\Gamma$ and $D \in \mathcal{C}_\Delta$, is in $\mathcal{L}[T]$. In case (Tid), the proof uses Proposition 5.4 (closure wrt substitution) and Proposition 5.11 (backward closure). In cases (T \otimes L), (T \rightarrow L), (Tcopy), (T \oplus L), (T $\&$ L₁), and (T $\&$ L₂), the proof proceeds in two steps: first, relying on Proposition 5.10 (forward closure) we show that every M'' such that $M \Longrightarrow M''$ is in $\mathcal{L}[T]$; then, we use this result in combination with Proposition 5.11 (backward closure) to conclude that $M \in \mathcal{L}[T]$. In cases (T1R), (T \otimes R), (T \rightarrow R), (T!R), (T \oplus R₁), and (T \oplus R₂), the proof consists in showing that M conforms to some specific case of Definition 5.4. Case (T1L) uses Proposition 5.12. Cases (T \otimes L), (T \rightarrow L), (T \oplus L), and (T $\&$ L₁), use the liveness guarantee given by Proposition 5.6. Cases (Tcopy), (T!L) and (Tcut¹) use Proposition 5.5 (closure under \equiv). Cases (Tcut), (T \rightarrow R), and (T!R) use Proposition 5.7 (closure under \equiv).

0. Case (Tid): $\Gamma; x:A \vdash [x \leftrightarrow z] :: z:A$.

Pick any $G \in \mathcal{C}_\Gamma$:

(a) $G \Downarrow, G \not\rightarrow$

[By Proposition 5.3]

(b) $D \in \mathcal{L}[x:A]$

(c) $M = (\mathbf{v}\tilde{u}, x)([x \leftrightarrow z] \mid G \mid D) \in \mathcal{L}[z:A]$

The proof of (c) is immediate:

- (d) $M \longrightarrow (\mathbf{v}\tilde{u})(G\{z/x\} \mid D\{z/x\}) \equiv! D\{z/x\} = M'$ [Since $x \notin \text{fn}(G)$]
- (e) $M' \in \mathcal{L}[z:A]$ [By (b) and Proposition 5.4]
- (f) $M \in \mathcal{L}[z:A]$ [By (d), (e), and Proposition 5.11]
- $[x \leftrightarrow z] \in \mathcal{L}[\Gamma; x:A \vdash z:A]$ [By (c) and Lemma 5.1]

1. Case (T1R): $\Gamma; \cdot \vdash \mathbf{0} :: z:\mathbf{1}$.

Pick any $G \in \mathcal{C}_\Gamma$:

- (a) $G \Downarrow, G \not\rightarrow$ [By Proposition 5.3]
- (b) $M = (\mathbf{v}\tilde{u})(\mathbf{0} \mid G) \in \mathcal{L}[z:\mathbf{1}]$

The proof of (b) is immediate:

- (c) $M \not\rightarrow \wedge M \equiv! \mathbf{0}$ [Using (a)]
- (d) $M \in \mathcal{L}[z:\mathbf{1}]$ [By (c) and Definition 5.4]
- $\mathbf{0} \in \mathcal{L}[\Gamma; \cdot \vdash z:\mathbf{1}]$ [By (b) and Lemma 5.1]

2. Case (T1L): $\Gamma; \Delta, z:\mathbf{1} \vdash P :: T$.

- (a) $\Gamma; \Delta \vdash P :: T$ [Premise of rule (T1L)]
- (b) $P \in \mathcal{L}[\Gamma; \Delta \vdash T]$ [By i.h. on (a)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

- (c) $M_1 = (\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}[T]$ [By Lemma 5.1 on (b)]
- Pick any $R \in \mathcal{L}[z:\mathbf{1}]$ and fix $M_2 = M_1 \mid R$
- (d) $M_2 \in \mathcal{L}[T]$ [By (c) and Proposition 5.12]
- (e) $(\mathbf{v}\tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid R) \in \mathcal{L}[T]$ [Expanding (d)]
- $P \in \mathcal{L}[\Gamma; \Delta, z:\mathbf{1} \vdash T]$ [By (e) and Lemma 5.1]

3. Case (T⊗L): $\Gamma; \Delta, z:A \otimes B \vdash z(y).P :: T$

- (a) $\Gamma; \Delta, y:A, z:B \vdash P :: T$ [Premise of rule (T⊗L)]
- (b) $P \in \mathcal{L}[\Gamma; \Delta, y:A, z:B \vdash T]$ [By i.h on (a)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

- (c) $G \Downarrow, G \not\rightarrow, D \Downarrow$ [By Proposition 5.3]
- (d) $(\mathbf{v}\tilde{u}, \tilde{x}, y, z)(P \mid G \mid D \mid \mathcal{L}[y:A] \mid \mathcal{L}[z:B]) \in \mathcal{L}[T]$ [By Lemma 5.1 on (b)]

Pick $R \in \mathcal{L}[z:A \otimes B]$:

- (e) $\cdot; \cdot \vdash R :: z:A \otimes B, R \Downarrow$ [By Definition 5.4]
- (f) $R \xrightarrow{\tilde{z}(y)} R'$ [By (e) and Proposition 5.6]
- (g) $R' \equiv! R'_1 \mid R'_2 \wedge R'_1 \in \mathcal{L}[y:A] \wedge R'_2 \in \mathcal{L}[z:B]$ [By Definition 5.4]

Fix $M = (\mathbf{v}\tilde{u}, \tilde{x}, z)(z(y).P \mid G \mid D \mid R)$

- (h) $\forall M''. M \Longrightarrow M'' \Rightarrow M'' \in \mathcal{L}[T]$

We prove (h) by induction on $k = \text{mlen}(D, R)$: [Possible by (c) and (e)]

Base case $k = 0$. Hence, $D \not\rightarrow$, and $R \not\rightarrow$:

$$M \longrightarrow (\mathbf{v}\tilde{u}, \tilde{x}, z, y)(P \mid G \mid D \mid R'_1 \mid R'_2) = M''$$

[Because of (f)]

$$M'' \in \mathcal{L}[T]$$

[Using (d) and (g)]

Inductive case $k > 0$:

Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$

$\forall M' \in W. M' \in \mathcal{L}[T]$

[By i.h.]

Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$

$\forall M'' \in W'. M'' \in \mathcal{L}[T]$

[By Proposition 5.10]

(i) $M \in \mathcal{L}[T]$

[By (h) and Proposition 5.11]

$z(y).P \in \mathcal{L}[\Gamma; \Delta, z:A \otimes B \vdash T]$

[By (i) and Lemma 5.1]

4. Case (T \otimes R): $\Gamma; \Delta, \Delta' \vdash \bar{z}(y).(P \mid Q) :: z:A \otimes B$

(a) $\Gamma; \Delta \vdash P :: y:A$

[Premise of rule (T \otimes R)]

(b) $\Gamma; \Delta' \vdash Q :: z:B$

[Premise of rule (T \otimes R)]

(c) $P \in \mathcal{L}[\Gamma; \Delta \vdash y:A]$

[By i.h on (a)]

(d) $Q \in \mathcal{L}[\Gamma; \Delta' \vdash z:B]$

[By i.h on (b)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta, D' \in \mathcal{C}_{\Delta'}$:

(e) $G \Downarrow, G \not\rightarrow, D \Downarrow, D' \Downarrow$

[By Proposition 5.3]

(f) $(\nu \tilde{u}, \tilde{x}_1)(P \mid G \mid D) \in \mathcal{L}[y:A]$

[By Lemma 5.1 on (c)]

(g) $(\nu \tilde{u}, \tilde{x}_2)(Q \mid G \mid D') \in \mathcal{L}[z:B]$

[By Lemma 5.1 on (d)]

Fix $\tilde{x} = \tilde{x}_1 \cup \tilde{x}_2$:

(h) $M = (\nu \tilde{u}, \tilde{x})(\bar{z}(y).(P \mid Q) \mid G \mid D \mid D') \in \mathcal{L}[z:A \otimes B]$

We prove (h) by induction on $k = \text{mlen}(D, D')$:

[Possible by (e)]

Base case $k = 0$. Hence, $D \not\rightarrow$, and $D' \not\rightarrow$:

(i) $M \xrightarrow{\bar{z}(y)} (\nu \tilde{u}, \tilde{x})(P \mid Q \mid D \mid D') = M'$

$M' \equiv: \underbrace{(\nu \tilde{u}, \tilde{x}_1)(P \mid G \mid D)}_{M'_1} \mid \underbrace{(\nu \tilde{u}, \tilde{x}_2)(Q \mid G \mid D')}_{M'_2}$

(j) $M'_1 \in \mathcal{L}[y:A]$

[By (f)]

(k) $M'_2 \in \mathcal{L}[z:B]$

[By (g)]

$M \in \mathcal{L}[z:A \otimes B]$

[By Definition 5.4, using (i), (j), and (k)]

Inductive case $k > 0$:

Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$

$\forall M' \in W. M' \in \mathcal{L}[z:A \otimes B]$

[By i.h.]

Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$

$\forall M'' \in W'. M'' \in \mathcal{L}[z:A \otimes B]$

[By Proposition 5.10]

$\bar{z}(y).(P \mid Q) \in \mathcal{L}[\Gamma; \Delta, \Delta' \vdash z:A \otimes B]$

[By (h) and Lemma 5.1]

5. Case (T \multimap L): $\Gamma; \Delta, \Delta', z:A \multimap B \vdash \bar{z}(y).(P \mid Q) :: T$

(a) $\Gamma; \Delta \vdash P :: y:A$

[Premise of rule (T \multimap L)]

(b) $\Gamma; \Delta', z:B \vdash Q :: T$

[Premise of rule (T \multimap L)]

(c) $P \in \mathcal{L}[\Gamma; \Delta \vdash y:A]$

[By i.h on (a)]

(d) $Q \in \mathcal{L}[\Gamma; \Delta', z:B \vdash T]$

[By i.h on (b)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta, D' \in \mathcal{C}_{\Delta'}$:

(e) $G \Downarrow, G \not\rightarrow, D \Downarrow, D' \Downarrow$

[By Proposition 5.3]

(f) $(\nu \tilde{u}, \tilde{x}_1)(P \mid G \mid D) \in \mathcal{L}[y:A]$

[By Lemma 5.1 on (c)]

- (g) $(\mathbf{v}\tilde{u}, \tilde{x}_2, z)(Q \mid G \mid D' \mid \mathcal{L}[z:B]) \in \mathcal{L}[T]$ [By Lemma 5.1 on (d)]
 Fix $\tilde{x} = \tilde{x}_1 \cup \tilde{x}_2$ and pick $R \in \mathcal{L}[z:A \multimap B]$:
 (h) $\cdot; \cdot \vdash R :: z:A \multimap B, R \Downarrow$ [By Definition 5.4]
 (i) $R \xrightarrow{z(y)} R'$ [By (h) and Proposition 5.6]
 (j) $\forall Q \in \mathcal{L}[y:A].(\mathbf{v}y)(R' \mid Q) \in \mathcal{L}[z:B]$ [By Definition 5.4]
 Fix $M = (\mathbf{v}\tilde{u}, \tilde{x}, z)(\bar{z}(y).(P \mid Q) \mid G \mid D \mid D' \mid R)$
 (k) $\forall M''. M \Longrightarrow M'' \Rightarrow M'' \in \mathcal{L}[T]$
 We prove (k) by induction on $k = \text{mlen}(D, D', R)$: [Possible by (e) and (h)]
 Base case $k = 0$. Hence, $D \not\rightarrow, D' \not\rightarrow, R \not\rightarrow$:
 $M \longrightarrow (\mathbf{v}\tilde{u}, \tilde{x}, z, y)(P \mid Q \mid G \mid D \mid D' \mid R') = M''$ [Because of (i)]
 Fix $M^* = (\mathbf{v}\tilde{u}, \tilde{x}_1)(P \mid G \mid D)$:
 (l) $M^* \in \mathcal{L}[y:A]$ [Using (f)]
 $M'' \equiv (\mathbf{v}\tilde{u}, \tilde{x}_2, z)(Q \mid G \mid D' \mid (\mathbf{v}y)(R' \mid M^*)) = M_1$
 (m) $(\mathbf{v}y)(R' \mid M^*) \in \mathcal{L}[z:B]$ [Using (j) and (l)]
 (n) $M_1 \in \mathcal{L}[T]$ [Using (g) and (m)]
 $M'' \in \mathcal{L}[T]$ [By Proposition 5.7 and (n)]
 Inductive case $k > 0$:
 Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$
 $\forall M' \in W. M' \in \mathcal{L}[T]$ [By i.h.]
 Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$
 $\forall M'' \in W'. M'' \in \mathcal{L}[T]$ [By Proposition 5.10]
 (o) $M \in \mathcal{L}[T]$ [By (k) and Proposition 5.11]
 $\bar{z}(y).(P \mid Q) \in \mathcal{L}[\Gamma; \Delta, \Delta', z:A \multimap B \vdash T]$ [By (o) and Lemma 5.1]

6. Case $(T \multimap R)$: $\Gamma; \Delta \vdash z(y).P :: z:A \multimap B$

- (a) $\Gamma; \Delta, y:A \vdash P :: z:B$ [Premise of rule $(T \multimap R)$]
 (b) $P \in \mathcal{L}[\Gamma; \Delta, y:A \vdash z:B]$ [By i.h on (a)]
 Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:
 (c) $G \Downarrow, G \not\rightarrow, D \Downarrow$ [By Proposition 5.3]
 (d) $(\mathbf{v}\tilde{u}, \tilde{x}, y)(P \mid G \mid D \mid \mathcal{L}[y:A]) \in \mathcal{L}[z:B]$ [By Lemma 5.1 on (b)]
 (e) $M = (\mathbf{v}\tilde{u}, \tilde{x})(z(y).P \mid G \mid D) \in \mathcal{L}[z:A \multimap B]$
 We prove (e) by induction on $k = \text{mlen}(D)$: [Possible by (c)]
 Base case $k = 0$. Hence, $D \not\rightarrow$:
 (f) $M \xrightarrow{z(y)} (\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D) = M_1$
 Pick any $R \in \mathcal{L}[y:A]$:
 (g) $(\mathbf{v}\tilde{u}, \tilde{x}, y)(P \mid G \mid D \mid R) \in \mathcal{L}[z:B]$ [Using (d)]
 $M \in \mathcal{L}[z:A \multimap B]$ [By Definition 5.4, using (f),(g)]
 Inductive case $k > 0$:
 Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$
 $\forall M' \in W. M' \in \mathcal{L}[z:A \multimap B]$ [By i.h.]
 Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$

$$\forall M'' \in W'. M'' \in \mathcal{L}[z:A \multimap B]$$

[Proposition 5.10]

$$z(y).P \in \mathcal{L}[\Gamma; \Delta \vdash z:A \multimap B]$$

[By Lemma 5.1 on (e)]

7. Case (Tcut): $\Gamma; \Delta, \Delta' \vdash (\nu z)(P \mid Q) :: T$

$$(a) \Gamma; \Delta \vdash P :: z:A$$

[Premise of rule (Tcut)]

$$(b) \Gamma; \Delta', z:A \vdash Q :: T$$

[Premise of rule (Tcut)]

$$(c) P \in \mathcal{L}[\Gamma; \Delta \vdash z:A]$$

[By i.h. on (a)]

$$(d) Q \in \mathcal{L}[\Gamma; \Delta', z:A \vdash T]$$

[By i.h. on (b)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta, D' \in \mathcal{C}_{\Delta'}$:

$$(e) (\nu \tilde{u}, \tilde{x}_1)(P \mid G \mid D) \in \mathcal{L}[z:A]$$

[By Lemma 5.1 on (c)]

$$(f) (\nu \tilde{u}, \tilde{x}_2, z)(Q \mid G \mid D' \mid \mathcal{L}[z:A]) \in \mathcal{L}[T]$$

[By Lemma 5.1 on (d)]

$$\text{Fix } M = (\nu \tilde{u}, \tilde{x})(\nu z)(P \mid Q) \mid G \mid D \mid D'$$

$$M \equiv_! (\nu z)((\nu \tilde{u}, \tilde{x}_2)(Q \mid G \mid D') \mid \underbrace{(\nu \tilde{u}, \tilde{x}_1)(P \mid G \mid D)}_{M_1}) = M'$$

$$(g) M_1 \in \mathcal{L}[z:A]$$

[Using (e)]

$$(h) M' \in \mathcal{L}[T]$$

[Combining (g) and (f)]

$$(i) M \in \mathcal{L}[T]$$

[Using (h) and Proposition 5.7]

$$(j) (\nu \tilde{u}, \tilde{x})(\nu z)(P \mid Q) \mid G \mid D \mid D' \in \mathcal{L}[T]$$

[Expanding (i)]

$$(\nu z)(P \mid Q) \in \mathcal{L}[\Gamma; \Delta, \Delta' \vdash T]$$

[By Lemma 5.1 on (j)]

8. Case (Tcut[!]): $\Gamma; \Delta \vdash (\nu z)(!z(y).P \mid Q) :: T$

$$(a) \Gamma; \cdot \vdash P :: y:A$$

[Premise of rule (Tcut[!])]

$$(b) \Gamma, z:A; \Delta \vdash Q :: T$$

[Premise of rule (Tcut[!])]

$$(c) P \in \mathcal{L}[\Gamma; \cdot \vdash y:A]$$

[By i.h. on (a)]

$$(d) Q \in \mathcal{L}[\Gamma, z:A; \Delta \vdash T]$$

[By i.h. on (b)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

$$(e) (\nu \tilde{u})(P \mid G) \in \mathcal{L}[y:A]$$

[By Lemma 5.1 on (c)]

$$(f) (\nu \tilde{u}, \tilde{x}, z)(Q \mid G \mid !z(y).\mathcal{L}[y:A] \mid D) \in \mathcal{L}[T]$$

[By Lemma 5.1 on (d)]

$$\text{Fix } M = (\nu \tilde{u}, \tilde{x})(!z(y).P \mid Q \mid G \mid D)$$

$$(g) M \in \mathcal{L}[T]$$

[Combining (e) and (f)]

$$M \equiv (\nu \tilde{u}, \tilde{x})(\nu z)(!z(y).P \mid Q) \mid G \mid D = M'$$

$$(h) M' \in \mathcal{L}[T]$$

[From (g), using Proposition 5.5]

$$(\nu z)(!z(y).P \mid Q) \in \mathcal{L}[\Gamma; \Delta \vdash T]$$

[By Lemma 5.1 on (h)]

9. Case (Tcopy): $\Gamma, z:A; \Delta \vdash \bar{z}(y).P :: T$

$$(a) \Gamma, z:A; \Delta, y:A \vdash P :: T$$

[Premise of rule (Tcopy)]

$$(b) P \in \mathcal{L}[\Gamma, z:A; \Delta, y:A \vdash P :: T]$$

[By i.h. on (a)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

$$(c) G \Downarrow, G \not\rightarrow, D \Downarrow$$

[By Proposition 5.3]

$$(d) (\nu \tilde{u}, z, \tilde{x}, y)(P \mid G \mid !z(y).\mathcal{L}[y:A] \mid D \mid \mathcal{L}[y:A]) \in \mathcal{L}[T]$$

[By Lemma 5.1 on (b)]

Pick $R \in \mathcal{L}[y:A]$:

$$\text{Fix } M = (\nu \tilde{u}, z, \tilde{x})(\bar{z}(y).P \mid G \mid !z(y).R \mid D)$$

(e) $\forall M''. M \Longrightarrow M'' \Rightarrow M'' \in \mathcal{L}[T]$

We prove (e) by induction on $k = \text{mlen}(D)$:

[Possible by (c)]

Base case $k = 0$. Hence, $D \not\rightarrow$:

$$M \longrightarrow \equiv (\nu \tilde{u}, z, \tilde{x}, y)(P \mid G \mid !z(y).R \mid D \mid R) = M''$$

$$M'' \in \mathcal{L}[T]$$

[Using (d) and Proposition 5.5]

Inductive case $k > 0$:

$$\text{Fix the set } W = \{M' \mid M \longrightarrow^{k-1} M'\}$$

$$\forall M' \in W. M' \in \mathcal{L}[T]$$

[By i.h.]

$$\text{Fix the set } W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$$

$$\forall M'' \in W'. M'' \in \mathcal{L}[T]$$

[By Proposition 5.10]

(f) $M \in \mathcal{L}[T]$

[By (e) and Proposition 5.11]

$$\bar{z}(y).P \in \mathcal{L}[\Gamma, z:A; \Delta \vdash T]$$

[By (f) and Lemma 5.1]

10. Case (T!L): $\Gamma; \Delta, y:!A \vdash P :: T$

$$(a) \Gamma, z:A; \Delta \vdash P\{z/y\} :: T$$

[Premise of rule (T!L)]

$$(b) P\{z/y\} \in \mathcal{L}[\Gamma, z:A; \Delta \vdash T]$$

[By i.h on (a)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

$$(c) (\nu \tilde{u}, z, \tilde{x})(P\{z/y\} \mid G \mid !z(w).\mathcal{L}[w:A] \mid D) \in \mathcal{L}[T]$$

[By Lemma 5.1 on (b)]

$$(d) (\nu \tilde{u}, y, \tilde{x})(P \mid G \mid \underbrace{!y(w).\mathcal{L}[w:A]}_R \mid D) \in \mathcal{L}[T]$$

[By $\equiv (\alpha\text{-conv})$ on (c)]

$$(e) R \in \mathcal{L}[y:!A]$$

[By Definition 5.4]

$$P \in \mathcal{L}[\Gamma; \Delta, y:!A \vdash T]$$

[By (d), (e), Proposition 5.5, and Lemma 5.1]

11. Case (T!R): $\Gamma; \cdot \vdash !z(y).P :: z:!A$

$$(a) \Gamma; \cdot \vdash P :: y:A$$

[Premise of rule (T!R)]

$$(b) P \in \mathcal{L}[\Gamma; \cdot \vdash y:A]$$

[By i.h on (a)]

Pick any $G \in \mathcal{C}_\Gamma$:

$$(c) G \Downarrow, G \not\rightarrow$$

[By Proposition 5.3]

$$(d) (\nu \tilde{u})(P \mid G) \in \mathcal{L}[y:A]$$

[By Lemma 5.1 on (b)]

$$\text{Fix } M = (\nu \tilde{u})(!z(y).P \mid G)$$

$$M \equiv !z(y).(\nu \tilde{u})(P \mid G) = M'$$

[By Definition 5.3, Axiom (2)]

$$(e) M' \in \mathcal{L}[z:!A]$$

[By Definition 5.4, using (d)]

$$(f) M \in \mathcal{L}[z:!A]$$

[By (e) and Proposition 5.7]

$$!z(y).P \in \mathcal{L}[\Gamma; \cdot \vdash z:!A]$$

[By (f) and Lemma 5.1]

12. Case (T⊕L): $\Gamma; \Delta, z:A \oplus B \vdash z.\text{case}(P, Q) :: T$

$$(a) \Gamma; \Delta, z:A \vdash P :: T$$

[Premise of rule (T⊕L)]

$$(b) \Gamma; \Delta, z:B \vdash Q :: T$$

[Premise of rule (T⊕L)]

$$(c) P \in \mathcal{L}[\Gamma; \Delta, z:A \vdash T]$$

[By i.h on (a)]

$$(d) Q \in \mathcal{L}[\Gamma; \Delta, z:B \vdash T]$$

[By i.h on (b)]

Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:

$$(e) G \Downarrow, G \not\rightarrow, D \Downarrow$$

[By Proposition 5.3]

$$(f) (\nu \tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid \mathcal{L}[z:A]) \in \mathcal{L}[T]$$

[By Lemma 5.1 on (c)]

- (g) $(\nu \tilde{u}, \tilde{x}, z)(Q \mid G \mid D \mid \mathcal{L}[z:B]) \in \mathcal{L}[T]$ [By Lemma 5.1 on (d)]
- Pick $R \in \mathcal{L}[z:A \oplus B]$:
- (h) $\cdot; \cdot \vdash R :: z:A \oplus B, R \Downarrow$ [By Definition 5.4]
- (i) $R \xrightarrow{\bar{z}.inl} R' \vee R \xrightarrow{\bar{z}.inr} R'$ [By (h) and Proposition 5.6]
- (j) $R \xrightarrow{\bar{z}.inl} R' \Rightarrow R' \in \mathcal{L}[z:A] \wedge R \xrightarrow{\bar{z}.inr} R' \Rightarrow R' \in \mathcal{L}[z:B]$ [By Definition 5.4]
- Fix $M = (\nu \tilde{u}, \tilde{x}, z)(z.\text{case}(P, Q) \mid G \mid D \mid R)$:
- (k) $\forall M''. M \Longrightarrow M'' \Rightarrow M'' \in \mathcal{L}[T]$
- We prove (k) by induction on $k = \text{mlen}(D, R)$: [Possible by (e) and (h)]
- Base case $k = 0$. Hence, $D \not\rightarrow, R \not\rightarrow$:
- $M \longrightarrow M'_1 \vee M'' \longrightarrow M'_2$, where:
- $M'_1 = (\nu \tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid R')$ [Because of (i)]
- $M'_2 = (\nu \tilde{u}, \tilde{x}, z)(Q \mid G \mid D \mid R')$ [Because of (i)]
- $M'_1 \in \mathcal{L}[T]$ [Using (f)]
- $M'_2 \in \mathcal{L}[T]$ [Using (g)]
- Inductive case $k > 0$:
- Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$
- $\forall M' \in W. M' \in \mathcal{L}[T]$ [By i.h.]
- Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$
- $\forall M'' \in W'. M'' \in \mathcal{L}[T]$ [By Proposition 5.10]
- (l) $M \in \mathcal{L}[T]$ [By (k) and Proposition 5.11]
- $z.\text{case}(P, Q) \in \mathcal{L}[\Gamma; \Delta, z:A \oplus B \vdash T]$ [By (l) and Lemma 5.1]

13. Case (T&L₁): $\Gamma; \Delta, z:A \& B \vdash z.inl; P :: T$

- (a) $\Gamma; \Delta, z:A \vdash P :: T$ [Premise of rule (T&L₁)]
- (b) $P \in \mathcal{L}[\Gamma; \Delta, z:A \vdash T]$ [By i.h on (a)]
- Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:
- (c) $G \Downarrow, G \not\rightarrow, D \Downarrow$ [By Proposition 5.3]
- (d) $(\nu \tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid \mathcal{L}[z:A]) \in \mathcal{L}[T]$ [By Lemma 5.1 on (b)]
- Pick $R \in \mathcal{L}[z:A \& B]$:
- (e) $\cdot; \cdot \vdash R :: z:A \& B, R \Downarrow$ [By Definition 5.4]
- (f) $R \xrightarrow{z.inl} R_1 \vee R \xrightarrow{z.inr} R_2$ [By (e) and Proposition 5.6]
- (g) $R \xrightarrow{z.inl} R_1 \Rightarrow R_1 \in \mathcal{L}[z:A] \wedge R \xrightarrow{z.inr} R_2 \Rightarrow R_2 \in \mathcal{L}[z:B]$ [By Definition 5.4]
- Fix $M = (\nu \tilde{u}, \tilde{x}, z)(z.inl; P \mid G \mid D \mid R)$:
- (h) $\forall M''. M \Longrightarrow M'' \Rightarrow M'' \in \mathcal{L}[T]$
- We prove (h) by induction on $k = \text{mlen}(D, R)$: [Possible by (c) and (e)]
- Base case $k = 0$. Hence, $D \not\rightarrow, R \not\rightarrow$:
- $M \longrightarrow (\nu \tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid R_1) = M''$
- $M'' \in \mathcal{L}[T]$ [Using (d) and (g)]
- Inductive case $k > 0$:
- Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$

$\forall M' \in W. M' \in \mathcal{L}[T]$ [By i.h.]
 Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$
 $\forall M'' \in W'. M'' \in \mathcal{L}[T]$ [By Proposition 5.10]
 (j) $M \in \mathcal{L}[T]$ [By (h) and Proposition 5.11]
 $z.\text{inl}; P \in \mathcal{L}[\Gamma; \Delta, z:A \& B \vdash T]$ [By (j) and Lemma 5.1]

14. Case (T&L₂): Analogous to case (T&L₁).

15. Case (T&R): $\Gamma; \Delta \vdash z.\text{case}(P, Q) :: z:A \& B$

(a) $\Gamma; \Delta \vdash P :: z:A$ [Premise of rule (T&R)]
 (b) $\Gamma; \Delta \vdash Q :: z:B$ [Premise of rule (T&R)]
 (c) $P \in \mathcal{L}[\Gamma; \Delta \vdash z:A]$ [By i.h on (a)]
 (d) $Q \in \mathcal{L}[\Gamma; \Delta \vdash z:B]$ [By i.h on (b)]
 Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:
 (e) $G \Downarrow, G \not\rightarrow, D \Downarrow$ [By Proposition 5.3]
 (f) $(\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}[z:A]$ [By Lemma 5.1 on (c)]
 (g) $(\mathbf{v}\tilde{u}, \tilde{x})(Q \mid G \mid D) \in \mathcal{L}[z:B]$ [By Lemma 5.1 on (d)]
 (h) $M = (\mathbf{v}\tilde{u}, \tilde{x})(z.\text{case}(P, Q) \mid G \mid D) \in \mathcal{L}[z:A \& B]$

We prove (h) by induction on $k = \text{mlen}(D)$: [Possible by (e)]

Base case $k = 0$. Hence, $D \not\rightarrow$:

(i) $M \xrightarrow{z.\text{inl}} M_1 \wedge M \xrightarrow{z.\text{inr}} M_2$, where :
 $M_1 = (\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D)$
 $M_2 = (\mathbf{v}\tilde{u}, \tilde{x})(Q \mid G \mid D)$
 (j) $M_1 \in \mathcal{L}[z:A]$ [Using (f)]
 (k) $M_2 \in \mathcal{L}[z:B]$ [Using (g)]
 $M \in \mathcal{L}[z:A \& B]$ [By Definition 5.4, using (i), (j), (k)]

Inductive case $k > 0$:

Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$
 $\forall M' \in W. M' \in \mathcal{L}[z:A \& B]$ [By i.h.]
 Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$
 $\forall M'' \in W'. M'' \in \mathcal{L}[z:A \& B]$ [By Proposition 5.10]
 $z.\text{case}(P, Q) \in \mathcal{L}[\Gamma; \Delta \vdash z:A \& B]$ [By (h) and Lemma 5.1]

16. Case (T⊕R₁): $\Gamma; \Delta \vdash z.\text{inl}; P :: z:A \oplus B$

(a) $\Gamma; \Delta \vdash z.\text{inl}; P :: z:A \oplus B$ [Premise of rule (T⊕R₁)]
 (b) $P \in \mathcal{L}[\Gamma; \Delta \vdash z:A]$ [By i.h on (a)]
 Pick any $G \in \mathcal{C}_\Gamma, D \in \mathcal{C}_\Delta$:
 (c) $G \Downarrow, G \not\rightarrow, D \Downarrow$ [By Proposition 5.3]
 (d) $(\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}[z:A]$ [By Lemma 5.1 on (b)]
 (e) $M = (\mathbf{v}\tilde{u}, \tilde{x})(z.\text{inl}; P \mid G \mid D) \in \mathcal{L}[z:A \oplus B]$

We prove (e) by induction on $k = \text{mlen}(D)$: [Possible by (c)]

Base case $k = 0$. Hence, $D \not\rightarrow$:

(i) $M \xrightarrow{z.\text{inl}} (\mathbf{v}\tilde{u}, \tilde{x})(P \mid G \mid D) = M_1$

(j) $M_1 \in \mathcal{L}[z:A]$	[Using (d)]
$M \in \mathcal{L}[z:A \oplus B]$	[Using (i), (j), and Definition 5.4]
Inductive case $k > 0$:	
Fix the set $W = \{M' \mid M \longrightarrow^{k-1} M'\}$	
$\forall M' \in W. M' \in \mathcal{L}[z:A \oplus B]$	[By i.h.]
Fix the set $W' = \{M'' \mid M' \longrightarrow M'' \wedge M' \in W\}$	
$\forall M'' \in W'. M'' \in \mathcal{L}[z:A \oplus B]$	[By Proposition 5.10]
$z.\text{inl}; P \in \mathcal{L}[\Gamma; \Delta \vdash z:A \oplus B]$	[By (e) and Lemma 5.1]

17. Case $(T \oplus R_2)$: Analogous to case $(T \oplus R_1)$. \square

A.5. Proof of Proposition 5.13

We repeat the statement in Page 267 and present its proof.

Proposition Appendix A.4 (Properties of confluent processes). Assume well-typed processes $P, P', P_1, \dots, P_k, Q$. Then we have:

1. Forward closure: If $P \diamond$ and $P \longrightarrow P'$ then $P' \diamond$.
2. Backward closure: If for all P_i such that $P \longrightarrow P_i$ we have that $P_i \diamond$, then $P \diamond$.
3. Closure wrt composition: Let P, Q be such that (i) $;\vdash P :: x:A$, (ii) $;\vdash Q :: x:A$, (iii) $P \diamond$, and (iv) $Q \diamond$. Then $(\nu x)(P \mid Q) \diamond$.

Proof. 1. By assumption, we have $P \diamond$ and $P \longrightarrow P'$. We have to show that for any P_1, P_2 such that $P' \Longrightarrow P_1$ and $P' \Longrightarrow P_2$, there exists process P_3 such that $P_1 \Longrightarrow P_3$ and $P_2 \Longrightarrow P_3$. Since $P \diamond$, for any R_1, R_2 such that $P \Longrightarrow R_1$ and $P \Longrightarrow R_2$, there exists an R' such that $R_1 \Longrightarrow R'$ and $R_2 \Longrightarrow R'$. This includes the particular case in which $P \longrightarrow P' \Longrightarrow P_1$ and $P \longrightarrow P' \Longrightarrow P_2$. The existence of a P_3 such that $P_1 \Longrightarrow P_3$ and $P_2 \Longrightarrow P_3$ follows from $P \diamond$. Therefore, $P' \diamond$.

2. Given that whenever $P \longrightarrow P_k$, we have $P_k \diamond$, we need to extend the diamond, starting from P . Precisely, we have to show that for any P_i, P_j such that $P \longrightarrow P_i, P \longrightarrow P_j, P_i \diamond$, and $P_j \diamond$, and for any R_1, R_2 such that $P \longrightarrow P_i \Longrightarrow R_1$ and $P \longrightarrow P_j \Longrightarrow R_2$, there exists an R' such that $R_1 \Longrightarrow R'$ and $R_2 \Longrightarrow R'$. The thesis follows by Proposition 5.8 (Page 266) which ensures that a reduction $P \longrightarrow P_i$ does not preclude all those reduction paths reachable if the first reduction is $P \longrightarrow P_j$. Since we have $P_i \diamond$ and $P_j \diamond$ by assumption, we may conclude that $P \diamond$, as desired.

3. There are two cases, depending on whether P and Q can interact. In turn, this depends on their typing.

- If $A = \mathbf{1}$ then P and Q do not interact and $P \mid Q$ represents their independent parallel composition. This guarantees that their reductions always proceed independently. Confluence of $P \mid Q$ can be inferred directly from that of P and Q , in a compositional way.
- If $A \neq \mathbf{1}$ then P and Q can interact. Well-typedness ensures that interactions occur exclusively on name x . We must show that for any R_1, R_2 such that

$$(\nu x)(P \mid Q) \Longrightarrow R_1$$

$$(\nu x)(P \mid Q) \Longrightarrow R_2$$

there is a process R such that $R_1 \Longrightarrow R$ and $R_2 \Longrightarrow R$. The only difference with respect to the previous case is that now weak transitions $(\nu x)(P \mid Q) \Longrightarrow R_i$ ($i \in \{1, 2\}$) consist of (a) reductions already present in (enabled from) P and Q , but also of (b) new reductions originated by the interaction of P and Q on x . Such new reductions also include those internal actions on (private) names different from x but occurring behind prefixes with subject x . The existence of R is ensured by the confluence of P and Q and the fact that new reductions originated from synchronizations on x are harmless, because (i) they are finite (Theorem 5.1); (ii) they are type preserving (Theorem 3.1); and, more importantly, by Proposition 5.8 (iii) these new reductions are not precluded by reductions enabled from P and Q , nor they preclude such enabled reductions. \square

A.6. Proof of Lemma 5.4

We repeat the statement in Page 269 and give details of the proof.

Lemma Appendix A.2. Let P be a process. If $\Gamma; \Delta \vdash P :: T$ then $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$.

Proof. By induction on the derivation of $\Gamma; \Delta \vdash P :: T$, with a case analysis on the last typing rule used. We have 18 cases to check; in all cases, we use [Lemma 5.3](#) to show that every $M = (\nu \tilde{u}, \tilde{x})(P \mid G \mid D)$ with $G \in \mathcal{C}_r^\diamond$ and $D \in \mathcal{C}_\Delta^\diamond$, is in $\mathcal{L}^\diamond[T]$.

The proof follows closely the lines of the proof of [Lemma 5.2](#) (Page 283); it exploits the fact that well-typed processes are always terminating ([Theorem 5.1](#)). In case (Tid), we use [Proposition 5.16](#) (closure wrt substitution) and [Proposition 5.20](#) (backward closure). In cases (T \otimes L), (T \multimap L), (Tcopy), (T \oplus L), (T $\&$ L₁), and (T $\&$ L₂), we proceed in two steps: first, using [Proposition 5.19](#) (forward closure) we show that every M'' such that $M \Rightarrow M''$ is in $\mathcal{L}^\diamond[T]$; then, we use this result in combination with [Proposition 5.20](#) (backward closure) to conclude that $M \in \mathcal{L}^\diamond[T]$. In cases (T1R), (T \otimes R), (T \multimap R), (T!R), (T \oplus R₁), and (T \oplus R₂), we show that M conforms to a specific case of [Definition 5.7](#). Case (T1L) uses [Proposition 5.13\(3\)](#). Cases (T \otimes L), (T \multimap L), (T \oplus L), and (T $\&$ L₁) use the liveness guarantee given by [Proposition 5.6](#). Cases (Tcopy), (T!L), and (Tcut¹) use [Proposition 5.17](#) (closure under \Rightarrow). Cases (Tcut), (T \multimap R), and (T!R) use [Proposition 5.18](#) (closure under $\equiv_!$).

Below, we illustrate a few cases; the rest are essentially as in the proof of [Lemma 5.2](#) (Page 283).

Case (Tid): $\Gamma; x:A \vdash [x \leftrightarrow z] :: z:A$.

Pick any $G \in \mathcal{C}_r^\diamond$:

(a) $G \diamond, G \not\rightarrow$ [By [Proposition 5.15](#)]

(b) $D \in \mathcal{L}^\diamond[x:A]$

(c) $M = (\nu \tilde{u}, x)([x \leftrightarrow z] \mid G \mid D) \in \mathcal{L}^\diamond[z:A]$

The proof of (c) is immediate:

(d) $M \longrightarrow (\nu \tilde{u})(G\{z/x\} \mid D\{z/x\}) \equiv_! D\{z/x\} = M'$ [Since $u_i \notin \text{fn}(D)$]

(e) $M' \in \mathcal{L}^\diamond[z:A]$ [By (b) and [Proposition 5.16](#)]

(f) $M \in \mathcal{L}^\diamond[z:A]$ [By (d), (e), and [Proposition 5.20](#)]

$[x \leftrightarrow z] \in \mathcal{L}^\diamond[\Gamma; x:A \vdash z:A]$ [By (c) and [Lemma 5.3](#)]

Case (T1R): $\Gamma; \cdot \vdash \mathbf{0} :: z:\mathbf{1}$.

Pick any $G \in \mathcal{C}_r^\diamond$:

(a) $G \diamond, G \not\rightarrow$ [By [Proposition 5.15](#)]

(b) $M = (\nu \tilde{u})(\mathbf{0} \mid G) \in \mathcal{L}^\diamond[z:\mathbf{1}]$

The proof of (b) is immediate:

(c) $M \diamond \wedge M \not\rightarrow \wedge M \equiv_! \mathbf{0}$ [Using (a)]

(d) $M \in \mathcal{L}^\diamond[z:\mathbf{1}]$ [By (c) and [Definition 5.7](#)]

$\mathbf{0} \in \mathcal{L}^\diamond[\Gamma; \cdot \vdash z:\mathbf{1}]$ [By (b) and [Lemma 5.3](#)]

Case (T1L): $\Gamma; \Delta, z:\mathbf{1} \vdash P :: T$.

(a) $\Gamma; \Delta \vdash P :: T$ [Premise of rule (T1L)]

(b) $P \in \mathcal{L}^\diamond[\Gamma; \Delta \vdash T]$ [By i.h. on (a)]

Pick any $G \in \mathcal{C}_r^\diamond, D \in \mathcal{C}_\Delta^\diamond$:

(c) $M_1 = (\nu \tilde{u}, \tilde{x})(P \mid G \mid D) \in \mathcal{L}^\diamond[T]$ [By [Lemma 5.3](#) on (b)]

Pick any $R \in \mathcal{L}^\diamond[z:\mathbf{1}]$ and fix $M_2 = M_1 \mid R$

(d) $M_2 \in \mathcal{L}^\diamond[T]$ [By (c) and [Proposition 5.13](#) (3)]

(e) $(\nu \tilde{u}, \tilde{x}, z)(P \mid G \mid D \mid R) \in \mathcal{L}^\diamond[T]$ [Expanding (d)]

$P \in \mathcal{L}^\diamond[\Gamma; \Delta, z:\mathbf{1} \vdash T]$ [By (e) and [Lemma 5.3](#)]. \square

Appendix B. Proofs from Section 6 (Typed context bisimilarity)

B.1. Proof of [Proposition 6.3](#)

We repeat the statement in [Page 273](#), and present its proof.

Proposition Appendix B.1 (*Proposition 6.3*). $\Gamma; \Delta \vdash P \approx Q :: T$ implies $\vdash K[P] \approx K[Q] :: T$, where K is any parallel representative in $\mathcal{K}_{\Gamma; \Delta \vdash T}$, as in *Definition 6.5*.

Proof. Let $\#(\Gamma)$, $\#(\Delta)$ denote the cardinality of Γ , Δ , respectively. The proof is by induction on $n = \#(\Gamma) + \#(\Delta)$. The base case is when $n = 0$: then both typing environments are empty and so $K = \bullet$. Hence, $K[P] = P$ and $K[Q] = Q$ and the thesis trivially holds. In the inductive case, $n > 0$, and there are two sub-cases. In the first one, we have $\Gamma, u_i:G_i; \Delta \vdash P \approx Q :: T$. By definition of \approx , it implies

$$\Gamma; \Delta \vdash (\nu u_i)(!u_i(y_i).S \mid P) \approx (\nu u_i)(!u_i(y_i).S \mid Q) :: T$$

for every S such that $\vdash S :: y_i:G_i$. Now, using the induction hypothesis, the latter allows us to infer $\vdash K_1[(\nu u_i)(!u_i(y_i).S \mid P)] \approx K_1[(\nu u_i)(!u_i(y_i).S \mid Q)] :: T$, for every $K_1 \in \mathcal{K}_{\Gamma, \Delta \vdash T}$. We observe that, for any R , $K_1[(\nu u_i)(!u_i(y_i).S \mid R)]$ is the same as having $K_0[R]$, with a context $K_0 = (\nu u_i)(!u_i(y_i).S \mid K_1[\cdot])$. By *Definition 6.5*, we infer that $K_0 \in \mathcal{K}_{\Gamma, u_i:G_i; \Delta \vdash T}$. Therefore, $\Gamma, u_i:G_i; \Delta \vdash P \approx Q :: T$ implies

$$\vdash K[P] \approx K[Q] :: T$$

for any $K \in \mathcal{K}_{\Gamma, u_i:G_i; \Delta}$, as desired. In the second sub-case, we have $\Gamma; \Delta, x_j:A_j \vdash P \approx Q :: T$, and the analysis follows the same lines as before. \square

B.2. Additional cases for proof of *Lemma 6.1*

We repeat the statement in Page 273, and detail some additional cases, thus complementing the proof given in that page.

Lemma Appendix B.1 (*Contextuality of \approx*). Typed context bisimilarity is a contextual relation, in the sense of *Definition 6.3*.

Proof. The proof proceeds by coinduction, exhibiting a typed context bisimulation for each of the conditions associated to *Definition 6.3*. We shall exploit the proof technique given by *Proposition 6.3*, which allows us to consider \approx under empty left-hand side contexts, for pairs of processes enclosed within appropriate parallel representatives. As a result, it suffices to consider only some of the conditions in *Figure 5*; see *Remark 6.1*. In Page 273 we have detailed the case of closure under output prefix; below we show the cases for closure under parallel composition and under replicated input (Items (8) and (15), respectively).

Item (8): We have to show that $\Gamma; \Delta_1 \vdash P \approx Q :: y:A$ implies

$$\Gamma; \Delta_1, \Delta_2 \vdash (\nu y)(P \mid S) \approx (\nu y)(Q \mid S) :: T$$

for any S, T such that $\Gamma; \Delta_2, y:A \vdash S :: T$. Using *Proposition 6.3*, it suffices to show $\vdash K_1[P] \approx K_1[Q] :: y:A$ implies

$$\vdash K_2[(\nu y)(K_1[P] \mid S)] \approx K_2[(\nu y)(K_1[Q] \mid S)] :: T$$

where $K_1 \in \mathcal{K}_{\Gamma; \Delta_1 \vdash y:A}$ and $K_2 \in \mathcal{K}_{\cdot; \Delta_2 \vdash x:T}$.

Letting $M = K_2[(\nu y)(K_1[P] \mid S)]$, $N = K_2[(\nu y)(K_1[Q] \mid S)]$ we show that

$$\mathcal{R}_8 = \{(M, N) : \vdash K_1[P] \approx K_1[Q] :: y:A, K_1 \in \mathcal{K}_{\Gamma; \Delta_1 \vdash y:A}, K_2 \in \mathcal{K}_{\cdot; \Delta_2 \vdash x:T}\} \cup \mathcal{W}_{\vdash T}$$

is a typed context bisimulation. Suppose that M moves first: $M \xrightarrow{\alpha} M'$; we need to find a matching action $N \xrightarrow{\alpha} N'$. Using the typing, we observe that there are two possibilities for α :

1. $\alpha = \tau$ and $\vdash M' :: T$, using subject reduction (*Theorem 3.1*)
2. $\alpha \neq \tau$, and both α and the type of M' depend on the actual shape of type T

We consider case (1) first, and so we assume that $M \xrightarrow{\tau} M'$. We examine the different possibilities for the origin of the reduction:

1. The reduction originates from K_2 . More precisely, by *Definition 6.5* the reduction originates in the part of K implementing names in Δ_1 , as the part of K_2 implementing names in Γ cannot evolve on its own (cf. *Definition 6.5*). Therefore, for some K_4 , we have both $K_2 \xrightarrow{\tau} K_4$ and $M' = K_4[(\nu y)(K_1[P] \mid S)]$. By subject reduction (*Theorem 3.1*), the type of K_4 is the same than that of K , which in turn implies $\vdash M' :: T$. Since K_2 occurs identically in M and N , this reduction can be matched by N , possibly preceded/followed by zero or more reductions: and so we have that $N \Rightarrow K_5[(\nu y)(K_3[Q'] \mid S')] = N'$, with $K_1 \Rightarrow K_3$, $K_2 \Rightarrow K_5$, $Q \Rightarrow Q'$, and $S \Rightarrow S'$. *Theorem 5.1* ensures that these weak transitions are finite. Moreover, subject reduction (*Theorem 3.1*) ensures $\vdash N' :: T$. Therefore, the pair (M', N') is in \mathcal{R}_8 , and we are done.

2. The reduction originates from K_1 . The argument proceeds analogously as in the previous case.
3. The reduction originates from P . Then, for some P' , we have $P \xrightarrow{\tau} P'$ and $M' = K_2[(\nu y)(K_1[P'] \mid S)]$. By subject reduction (Theorem 3.1), the type remains unchanged, which in turn implies $\vdash M' :: T$. Since $\vdash K_1[P] \approx K_1[Q] :: y:A$, we infer that N can match this reduction: there is a Q' such that $Q \Longrightarrow Q'$. Again, reductions from Q may be preceded or followed by reductions from K_1 , K_2 , and S . More precisely, there is a weak transition $N \Longrightarrow K_5[(\nu y)(K_3[Q'] \mid S')] = N'$, with $K_1 \Longrightarrow K_3$, $K_2 \Longrightarrow K_5$, $Q \Longrightarrow Q'$, and $S \Longrightarrow S'$. Theorem 5.1 ensures these weak transitions are finite. Moreover, subject reduction (Theorem 3.1) ensures $\vdash N' :: T$. Therefore, the pair (M', N') is in \mathcal{R}_8 , and we are done.
4. The reduction originates from S . We proceed analogously as in the previous cases, relying on the fact that S is the same in M and N .
5. The reduction originates from the interaction of P and K_1 . Therefore, for some K_3, P' , we have $M' = K_2[(\nu y)(K_3[P'] \mid S)]$. By subject reduction (Theorem 3.1), we can infer that $\vdash M' :: T$. Since K_1 occurs identically in M and N , and $\vdash K_1[P] \approx K_1[Q] :: y:A$, we infer that this interaction can be matched by N . Hence, there is a weak transition $N \Longrightarrow K_5[(\nu y)(K_3[Q'] \mid S')] = N'$ with $K_1 \Longrightarrow K_3$, $K_2 \Longrightarrow K_5$, $Q \Longrightarrow Q'$, and $S \Longrightarrow S'$. Theorem 5.1 ensures these weak transitions are finite. Moreover, subject reduction (Theorem 3.1) ensures $\vdash N' :: T$. Therefore, the pair (M', N') is in \mathcal{R}_8 , and we are done.
6. The reduction originates from the interaction of S and K_2 . The argument proceeds analogously as in the previous case.
7. The reduction originates from the interaction of P and S . Therefore, $M' = K_2[(\nu y)(K_1[P'] \mid S')]$. Using the typings of each process, we infer that this interaction is only possible via a synchronization on y , which offers (case of P) and requires (case of S) a behavior described by A . We then proceed by structural induction on type A . All the cases are covered by preservation lemmas which formalize the interaction of complementary actions. We detail only the case $A = A_1 \otimes A_2$; the other cases are similar. Using Lemma 3.2 we infer $P \xrightarrow{\bar{x}(y)} P'$ and $S \xrightarrow{x(y)} S'$. Using Lemma 3.3 we infer that P' is well-typed, and we have $\vdash K_2[(\nu y)(K_1[P'] \mid S')] :: T$. Since $\vdash K_1[P] \approx K_1[Q] :: y:A$ and S is the same in N , we know that these actions can be matched by N , and that there exist Q', S' such that $Q \xrightarrow{\bar{x}(y)} Q'$ and $S \xrightarrow{x(y)} S'$. Hence, there is an $N' = K_5[(\nu y)(K_3[Q'] \mid S')]$ with $K_1 \Longrightarrow K_3$ and $K_2 \Longrightarrow K_5$. By virtue of Theorem 5.1 these are all finite weak transitions. Using again Lemma 3.3 and subject reduction (Theorem 3.1), one can show that N' is well-typed: $\vdash K_5[(\nu y)(K_3[Q'] \mid S')] :: T$. Therefore, the pair (M', N') is in \mathcal{R}_8 and we are done.

Now we consider case (2), and so we assume $M \xrightarrow{\alpha} M'$, for some $\alpha \neq \tau$. The shape of α depends on the structure of T ; the typing information ensures that T can only be provided by S . Therefore, we proceed by induction on the structure of T . We consider only the case $T = x:A_1 \otimes A_2$; the other cases are similar or simpler. Then, by Lemma 3.2, $\alpha = \bar{x}(z)$ and $M' = K_2[(\nu y)(K_1[P] \mid S')]$. Since S is the same in N , we know that this action can be matched by N : indeed we have $S \xrightarrow{\bar{x}(z)} S'$ and $N' = K_5[(\nu y)(K_3[Q'] \mid S')]$, with $K_1 \Longrightarrow K_3$, $K_2 \Longrightarrow K_5$, $Q \Longrightarrow Q'$, and $S \Longrightarrow S'$. Theorem 5.1 ensures these weak transitions are finite. Now we follow the definition of \approx for output actions. Then, for any R such that $z:A_1 \vdash R :: -:\mathbf{1}$, we verify that both $\vdash (\nu z)(M' \mid R) :: x:A_2$ and $\vdash (\nu z)(N' \mid R) :: x:A_2$ hold. Hence, the pair $((\nu z)(M' \mid R), (\nu z)(N' \mid R))$ is in $\mathcal{W}_{\vdash x:A_1 \otimes A_2}$ and we are done.

The case in which $N \xrightarrow{\alpha} N'$ moves first is completely symmetric.

Item (15): We have to show that $\Gamma; \Delta \vdash P \approx Q :: y:A$ implies

$$\Gamma; \Delta \vdash !x(y).P \approx !x(y).Q :: x:A$$

Using Proposition 6.3, it suffices to show that $\vdash K[P] \approx K[Q] :: y:A$ implies

$$\vdash !x(y).K[P] \approx !x(y).K[Q] :: x:A$$

for any $K \in \mathcal{K}_{\Gamma; \Delta \vdash y:A}$. Let $M = !x(y).K[P]$ and $N = !x(y).K[Q]$. We show that

$$\mathcal{R}_{13} = \{(M, N) : \vdash K[P] \approx K[Q] :: y:A, K \in \mathcal{K}_{\Gamma; \Delta \vdash T}\} \cup \mathcal{W}_{\vdash x:A}$$

is a typed context bisimulation. Suppose M moves first: $M \xrightarrow{\alpha} M'$. We must find a matching action from N such that $N \xrightarrow{\alpha} N'$. The only possibility is an input on x and so we have $M \xrightarrow{x(z)} !x(y).K[P] \mid K[P]\{z/y\} = M'$. Process N can match this action immediately: $N \xrightarrow{x(z)} !x(y).K[Q] \mid K[Q]\{z/y\} = N'$. It is easy to show that typing is preserved by substitution, and so $\vdash K[P] \approx K[Q] :: y:A$ allows to infer $\vdash K[P]\{z/y\} \approx K[Q]\{z/y\} :: z:A$.

Following the clause for replicated input of \approx , we consider the closure of M' and N' with a process L such that $z:A \vdash L :: -:\mathbf{1}$. Such closures correspond, respectively, to

$$(\nu z)(K[P]\{z/y\} \mid L) \mid !x(y).K[P] \quad \text{and} \quad (\nu z)(K[Q]\{z/y\} \mid L) \mid !x(y).K[Q]$$

We verify the type of these closures is indeed $x:A$, as required by the replicated input clause. Since $\vdash K[P]\{z/y\}, K[Q]\{z/y\} :: z:A$, these processes can be composed with L , thus leading to processes of type $-:\mathbf{1}$.

It is immediate to see that $\vdash !x(y).K[P], !x(y).K[Q] :: x:A$; hence, via an independent parallel composition the two processes above are of type $x:A$, and the pair

$$((\nu z)(K[P]\{z/y\} \mid L) \mid !x(y).K[P], (\nu z)(K[Q]\{z/y\} \mid L) \mid !x(y).K[Q])$$

is in \mathcal{R}_{13} , as desired. The reasoning when N moves first is completely symmetric. \square

Appendix C. Proofs from Section 7.1 (Applications)

C.1. Additional cases for the proof of Theorem 7.1

We repeat the statement in Page 275, and detail some additional cases, thus complementing the proof given in that page.

Theorem Appendix C.1 (Theorem 7.1). *Let P, Q be processes such that*

- (i) $\Gamma; \Delta \vdash D \leadsto P :: T$;
- (ii) $\Gamma; \Delta \vdash E \leadsto Q :: T$;
- (iii) $P \simeq_c Q$.

Then, $\Gamma; \Delta \vdash P \approx Q :: T$.

Proof. By coinduction, exhibiting appropriate typed context bisimulations for each commuting conversion. In the bisimulation game, we exploit termination of well-typed processes (Theorem 5.1) to ensure that actions can be matched with finite weak transitions, and Theorem 3.1 to ensure preservation of type under reductions. We detail the cases of proof conversions A-2 and A-4 (cf. Fig. D.6), and C-11 (cf. Fig. D.7).

Proof conversion A-2 We then have that

$$\begin{aligned} \Gamma; \Delta \vdash \text{cut}D(x.\otimes RE_x F) \leadsto M &= (\nu x)(\hat{D} \mid \bar{z}(y).(\hat{E} \mid \hat{F})) :: z:A \otimes B \\ \Gamma; \Delta \vdash \otimes R(\text{cut}D(x.E_x))F \leadsto N &= \bar{z}(y).((\nu x)(\hat{D} \mid \hat{E}) \mid \hat{F}) :: z:A \otimes B \end{aligned}$$

with

$$\Gamma; \Delta_1 \vdash \hat{D} :: x:C \quad \Gamma; \Delta_2, x:C \vdash \hat{E} :: y:A \quad \Gamma; \Delta_3 \vdash \hat{F} :: z:B \tag{C.1}$$

and $\Delta = \Delta_1, \Delta_2, \Delta_3$. We show that $M \simeq_c N$ implies $\Gamma; \Delta \vdash M \approx N :: z:A \otimes B$.

By virtue of Proposition 6.3, we have to show that for every $K \in \mathcal{K}_{\Gamma; \Delta}$, we have $\vdash \cdot \vdash K[M] \approx K[N] :: z:A \otimes B$. In turn, this implies exhibiting a typed context bisimilarity \mathcal{R} containing the pair $(K[M], K[N])$. We thus define \mathcal{R} as :

$$\begin{aligned} \mathcal{R} &= \mathcal{W}_{\vdash z:A \otimes B} \cup \mathcal{S} \cup \mathcal{S}^{-1} \quad \text{where:} \\ \mathcal{S} &= \{(K_1[M'], K_2[N]) : M \Longrightarrow M', K_1, K_2 \in \mathcal{K}_{\Gamma; \Delta}\} \end{aligned}$$

and $\mathcal{W}_{\vdash z:A \otimes B}$ is as in Definition 6.6. Notice that \mathcal{S} is a type-respecting relation indexed by $\vdash z:A \otimes B$. In fact, using the typings in (C.1)—with $\Gamma = \Delta = \emptyset$ —and exploiting subject reduction (Theorem 3.1), it can be checked that for all $(P, Q) \in \mathcal{S}$ both $\vdash P :: z:A \otimes B$ and $\vdash Q :: z:A \otimes B$ can be derived.

We now show that \mathcal{R} is a typed context bisimilarity. Pick any $K \in \mathcal{K}_{\Gamma; \Delta}$. Using Definition 6.5, we can assume

$$K = (\nu \tilde{u}, \tilde{x})(\bullet \mid K_\Gamma \mid K_\Delta) \quad \text{where:}$$

- $K_\Gamma = \prod_{i \in I} !u_i(y_i).R_i$, with $\vdash R_i :: y_i:D_i$, for every $u_i:D_i \in \Gamma$;
- $K_\Delta = \prod_{j \in J} S_j$, with $\vdash S_j :: x_j:C_j$, for every $x_j:C_j \in \Delta$.

Clearly, $(K[M], K[N]) \in \mathcal{S}$, and so it is in \mathcal{R} . Now, suppose $K[M]$ moves first: $K[M] \xrightarrow{\alpha} M_1^*$. We have to find a matching action α from $K[N]$, i.e., $K[N] \xrightarrow{\alpha} N_1^*$. Since $\vdash K[M] :: z:A \otimes B$, we have two possible cases for α :

1. Case $\alpha = \tau$. We consider the possibilities for the origin of the reduction:
 - (a) $K_\Gamma \xrightarrow{\tau} K'_\Gamma$ and $K[M] \xrightarrow{\tau} K'[M]$. However, this cannot be the case, as by construction K_Γ corresponds to the parallel composition of input-guarded replicated processes which cannot evolve on their own.
 - (b) $K_\Delta \xrightarrow{\tau} K'_\Delta$ and $K[M] \xrightarrow{\tau} K'[M]$. Then, for some $l \in J$, $S_l \xrightarrow{\tau} S'_l$:

$$K[M] \xrightarrow{\tau} (\nu \tilde{u}, \tilde{x})(K_\Gamma \mid K'_\Delta \mid M) = K'[M] = M_1^*$$

Now, context K is the same in $K[N]$. Then K_Δ occurs identically in $K[N]$, and this reduction can be matched by a finite weak transition:

$$K[N] \Longrightarrow (\nu \tilde{u}, \tilde{x})(K_\Gamma \mid K''_\Delta \mid N) = K''[N] = N_1^*$$

By subject reduction ([Theorem 3.1](#)), $\vdash S'_i :: x_i : C_i$; hence, K', K'' are in $\mathcal{K}_{\Gamma, \Delta}$. Hence, the pair $(K'[M], K''[N])$ is in \mathcal{S} (as $M \Longrightarrow M$) and so it is in \mathcal{R} .

- (c) $M \xrightarrow{\tau} M'$ and $K[M] \xrightarrow{\tau} K[M']$. Since $M = (\nu x)(\hat{D} \mid \bar{z}(y).(\hat{E} \mid \hat{F}))$, the only possibility is that there is a \hat{D}_1 such that $\hat{D} \xrightarrow{\tau} \hat{D}_1$ and $M' = (\nu x)(\hat{D}_1 \mid \bar{z}(y).(\hat{E} \mid \hat{F}))$. This way,

$$K[M] \xrightarrow{\tau} (\nu \tilde{u}, \tilde{x})(K_\Gamma \mid K_\Delta \mid M') = K[M'] = M_1^*$$

We observe that $K[N]$ cannot match this action, but $K[N] \Longrightarrow K[N]$ is a valid weak transition. Hence, $N_1^* = K[N]$. By subject reduction ([Theorem 3.1](#)), we infer that $\vdash K[M'] :: z : A \otimes B$. We use this fact to observe that the pair $(K[M'], K[N])$ is included in \mathcal{S} . Hence, it is in \mathcal{R} .

- (d) There is an interaction between M and K_Γ or between M and K_Δ : this is only possible by the interaction of \hat{D} with K_Γ or K_Δ on names in \tilde{u}, \tilde{x} . Again, the only possible weak transition from $K[N]$ matching this reduction is $K[N] \Longrightarrow K[N]$, and the analysis proceeds as in the previous case.
2. Case $\alpha \neq \tau$. Then the only possibility, starting from $K[M]$, is an output action of the form $\alpha = \bar{z}(y)$. This action can only originate in M :

$$K[M] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K_\Gamma \mid K_\Delta \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid \hat{F}))) = M_1^*$$

Process $K[N]$ can match this action via the following finite (weak) transition:

$$K[N] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}') \mid \hat{F}')) = N_1^*$$

Observe how N_1^* reflects the changes in $K[N]$ due to the possible reductions before and after α . By definition of \approx (output case), we consider the composition of M_1^* and N_1^* with any V such that $y : A \vdash V :: - : \mathbf{1}$. Using the typings in ([C.1](#)) and subject reduction ([Theorem 3.1](#)), we infer

$$\begin{aligned} \vdash M_2^* &= (\nu \tilde{x}, \tilde{u})(K_\Gamma \mid K_\Delta \mid (\nu x)(\hat{D} \mid (\nu y)(\hat{E} \mid V \mid \hat{F}))) :: z : B \\ \vdash N_2^* &= (\nu \tilde{x}, \tilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu y)((\nu x)(\hat{D}' \mid \hat{E}' \mid V) \mid \hat{F}')) :: z : B \end{aligned}$$

Hence, the pair (M_2^*, N_2^*) is in $\mathcal{W}_{\vdash z : A \otimes B}$ and so it is in \mathcal{R} .

Now, let us suppose that $K[N]$ moves first: $K[N] \xrightarrow{\alpha} N_1^*$. We have to find a matching action α from $K[M]$: $K[M] \xrightarrow{\alpha} M_1^*$. Similarly as before, there are two cases: either $\alpha = \tau$ or $\alpha = \bar{z}(y)$. The former is as detailed before; the only difference is that reductions from $K[N]$ can only be originated in K_Δ ; these are matched by $K[M]$ with weak transitions originating in both K and in M . We therefore obtain pairs of processes in \mathcal{S}^{-1} .

We now detail the case in which $\alpha = \bar{z}(y)$. We have:

$$K[N] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K_\Gamma \mid K_\Delta \mid (\nu y)((\nu x)(\hat{D} \mid \hat{E}) \mid \hat{F})) = N_1^*$$

and this action can be matched by $K[M]$ with a finite weak transition:

$$K[M] \xrightarrow{\bar{z}(y)} (\nu \tilde{x}, \tilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu x)(\hat{D}' \mid (\nu y)(\hat{E}' \mid \hat{F}')))) = M_1^*$$

where M_1^* takes into account the possible reductions before and after α . As before, we consider the composition of N_1^* and M_1^* with any V such that $y : A \vdash V :: - : \mathbf{1}$. Using ([C.1](#)), we can infer both

$$\begin{aligned} \vdash N_2^* &= (\nu \tilde{x}, \tilde{u})(K_\Gamma \mid K_\Delta \mid (\nu y)((\nu x)(\hat{D} \mid \hat{E} \mid V) \mid \hat{F})) :: z : B \\ \vdash M_2^* &= (\nu \tilde{x}, \tilde{u})(K'_\Gamma \mid K'_\Delta \mid (\nu x)(\hat{D}' \mid (\nu y)(\hat{E}' \mid V \mid \hat{F}')))) :: z : B \end{aligned}$$

Hence, the pair (N_2^*, M_2^*) is in $\mathcal{W}_{\vdash z : A \otimes B}$ and so it is in \mathcal{R} . This concludes the proof for this case.

Proof conversion A-4 We then have that

$$\begin{aligned} \Gamma; \Delta, y : A \otimes B \vdash \text{cut} D(x. \otimes \text{Ly}(z.y. E_{xzy})) \rightsquigarrow M &= (\nu x)(\hat{D} \mid y(z). \hat{E}) :: T \\ \Gamma; \Delta, y : A \otimes B \vdash \otimes \text{Ly}(z.y. \text{cut} D(x. E_{xzy})) \rightsquigarrow N &= y(z). (\nu x)(\hat{D} \mid \hat{E}) :: T \end{aligned}$$

with

$$\Gamma; \Delta_1 \vdash \hat{D} :: x : C \quad \Gamma; \Delta_2, x : C, z : A, y : B \vdash \hat{E} :: T \quad (\text{C.2})$$

and $\Delta = \Delta_1, \Delta_2$. We show that $M \simeq_c N$ implies $\Gamma; \Delta, y : A \otimes B \vdash M \approx N :: T$.

By virtue of [Proposition 6.3](#), we have to show that for every $K \in \mathcal{K}_{\Gamma, \Delta, y : A \otimes B}$, we have $\cdot; \vdash K[M] \approx K[N] :: T$. In turn, this implies exhibiting a typed context bisimilarity \mathcal{R} containing the pair $(K[M], K[N])$. We thus define \mathcal{R} as

$$\mathcal{R} = \mathcal{I}_{\vdash T} \cup \mathcal{W}_{\vdash T}$$

recalling that $\mathcal{I}_{\Gamma; \Delta \vdash T}$ stands for the relation $\{(P, Q) : \Gamma; \Delta \vdash P :: T, \Gamma; \Delta \vdash Q :: T\}$. We show that \mathcal{R} is a typed context bisimilarity. Pick any $K \in \mathcal{K}_{\Gamma; \Delta, y:A \otimes B}$. Using Definition 6.5, we can assume

$$K = (\mathbf{v}\tilde{u}, \tilde{x}, y)(\bullet \mid K_{\Gamma} \mid K_{\Delta} \mid V)$$

where

- $K_{\Gamma} \equiv \prod_{i \in I} !u_i(y_i).R_i$, with $\vdash R_i :: y_i : G_i$, for every $u_i : G_i \in \Gamma$;
- $K_{\Delta} \equiv \prod_{j \in J} S_j$, with $\vdash S_j :: x_j : C_j$, for every $x_j : C_j \in \Delta$;
- $\vdash V :: y : A \otimes B$.

Clearly, $(K[M], K[N]) \in \mathcal{L}_T$, and so it is in \mathcal{R} . Now, suppose $K[M]$ moves first: $K[M] \xrightarrow{\alpha} M_1^*$. We have to find a matching action α from $K[N]$, i.e., $K[N] \xRightarrow{\alpha} N_1^*$. We consider two possible cases:

1. Case $\alpha = \tau$. We consider the possibilities for the origin of the reduction:

- (a) $K_{\Gamma} \xrightarrow{\tau} K'_{\Gamma}$: This cannot be the case, as by construction this process corresponds to the composition of zero or more input-guarded replications which cannot evolve on their own.
- (b) $K_{\Delta} \xrightarrow{\tau} K'_{\Delta}$ and $K[M] \xrightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x}, y)(K_{\Gamma} \mid K'_{\Delta} \mid V \mid M) = M_1^*$. Since K_{Δ} occurs identically in both processes, this reduction can be matched by $K[N]$ with a finite weak transition:

$$K[N] \xRightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x}, y)(K_{\Gamma} \mid K''_{\Delta} \mid V' \mid M') = N_1^*$$

Using subject reduction (Theorem 3.1) it can be shown that $K', K'' \in \mathcal{K}_{\Gamma; \Delta, y:A \otimes B}$, and that V' and M' preserve the type of V and M , respectively. Hence, both $\vdash M_1^* :: T$ and $\vdash N_1^* :: T$ hold, and the pair (M_1^*, N_1^*) is in \mathcal{L}_T and so it is in \mathcal{R} .

- (c) $V \xrightarrow{\tau} V'$ and $K[M] \xrightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x}, y)(K_{\Gamma} \mid K_{\Delta} \mid V' \mid M) = M_1^*$. This case proceeds similarly as the previous one, as V occurs in both processes.
- (d) $M \xrightarrow{\tau} M'$ and $K[M] \xrightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid V \mid M') = M_1^*$. Since $M = (\mathbf{v}x)(\hat{D} \mid y(z).\hat{E})$, the only possibility is that there is a \hat{D}_1 such that $\hat{D} \xrightarrow{\tau} \hat{D}_1$ and $M' = (\mathbf{v}x)(\hat{D}_1 \mid y(z).\hat{E})$. This way,

$$K[M] \xrightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid V \mid M') = K[M'] = M_1^*$$

We observe that $K[N]$ cannot match this action, as \hat{D} is behind a prefix. Nevertheless, $K[N] \xRightarrow{\tau} K[N]$ is a valid weak transition, and so $N_1^* = K[N]$. By subject reduction (Theorem 3.1), we infer that $\vdash K[M'] :: T$. Hence, the pair (M_1^*, N_1^*) is included in \mathcal{L}_T , and so it is in \mathcal{R} .

- (e) The reduction arises from the interaction of V and M . This can only correspond to a synchronization on y . We have:

$$K[M] \xrightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid (\mathbf{v}y)(V' \mid (\mathbf{v}x)(\hat{D} \mid \hat{E}\sigma))) = M_1^*$$

where σ stands for the substitution derived from the synchronization. This reduction can be matched by $K[N]$ via a finite weak transition:

$$K[N] \xRightarrow{\tau} (\mathbf{v}\tilde{u}, \tilde{x})(K_{\Gamma} \mid K'_{\Delta} \mid (\mathbf{v}y)(V' \mid (\mathbf{v}x)(\hat{D}' \mid \hat{E}')\sigma)) = N_1^*$$

where N_1^* captures the fact that internal actions could have occurred before and after the synchronization on y . By subject reduction (Theorem 3.1), typing is preserved in both cases, and so $(M_1^*, N_1^*) \in \mathcal{R}$.

2. Case $\alpha \neq \tau$. Then α corresponds to the execution of some behavior described by T , in the right-hand side typing. However, this cannot be the case since, as specified by the typings in (C.2), the behavior described by T can only be provided by \hat{E} , which is behind an input prefix on y , both in $K[M]$ and $K[N]$. Therefore, behavior described by T cannot be exercised until such a prefix is consumed, and we have that, necessarily, $\alpha = \tau$. Observe that once such prefixes are consumed (via internal actions) the evolution corresponding to the behavior described by T is still in \mathcal{R} , as the continuation relation $\mathcal{W}_{\vdash T}$ is in \mathcal{R} .

The analysis when $K[N]$ moves first follows the same lines and is omitted.

Proof conversion C-11 We then have that

$$\begin{aligned} \Gamma; \Delta, y:A \oplus B \vdash \text{cut}^1 D(u.\oplus \text{Ly}(y.E_{uy})(y.F_{uy})) &\rightsquigarrow M = (\mathbf{v}u)((!u(z).\hat{D}) \mid y.\text{case}(\hat{E}, \hat{F})) :: T \\ \Gamma; \Delta, y:A \oplus B \vdash \oplus \text{Ly}(y.\text{cut}^1 D(u.E_{uy}))(y.\text{cut}^1 D(u.F_{uy})) & \\ \rightsquigarrow N = y.\text{case}((\mathbf{v}u)((!u(z).\hat{D}) \mid \hat{E}), (\mathbf{v}u)((!u(z).\hat{D}) \mid \hat{F})) &:: T \end{aligned}$$

with

$$\Gamma; \cdot \vdash \hat{D} :: z:C \quad \Gamma, u:C; \Delta_1, y:A \vdash \hat{E} :: T \quad \Gamma, u:C; \Delta_2, y:B \vdash \hat{F} :: T \quad (\text{C.3})$$

and $\Delta = \Delta_1, \Delta_2$. We show that $M \simeq_c N$ implies $\Gamma; \Delta \vdash M \approx N :: T$.

By virtue of [Proposition 6.3](#), we have to show that for every $K \in \mathcal{K}_{\Gamma; \Delta, y: A \oplus B}$, we have $\cdot; \vdash K[M] \approx K[N] :: T$. In turn, this implies exhibiting a typed context bisimilarity \mathcal{R} containing the pair $(K[M], K[N])$. We thus define \mathcal{R} as

$$\mathcal{R} = \mathcal{I}_{\vdash T} \cup \mathcal{W}_{\vdash T}$$

We now show that \mathcal{R} is a typed context bisimilarity. Pick any $K \in \mathcal{K}_{\Gamma; \Delta, y: A \oplus B}$. Using [Definition 6.5](#), we can assume

$$K = (\nu \tilde{u}, \tilde{x}, y)(\bullet \mid K_{\Gamma} \mid K_{\Delta} \mid V)$$

where

- $K_{\Gamma} \equiv \prod_{i \in I} !u_i(y_i).R_i$, with $\vdash R_i :: y_i:D_i$, for every $u_i:D_i \in \Gamma$;
- $K_{\Delta} \equiv \prod_{j \in J} S_j$, with $\vdash S_j :: x_j:C_j$, for every $x_j:C_j \in \Delta$;
- $\vdash V :: y:A \oplus B$.

Clearly, $(K[M], K[N]) \in \mathcal{R}$. Now, suppose $K[M]$ moves first: $K[M] \xrightarrow{\alpha} M_1^*$. We have to find a matching action α from $K[N]$, i.e., $K[N] \xrightarrow{\alpha} N_1^*$. The analysis is similar to the one detailed for the commuting conversion No. A-4. We consider two possible cases:

1. Case $\alpha = \tau$. We consider the possibilities for the origin of the reduction:

- (a) $K_{\Gamma} \longrightarrow K'_{\Gamma}$: This cannot be the case, as by construction this process corresponds to the composition of zero or more input guarded replications which cannot evolve on their own.
- (b) $M \longrightarrow M'$: This cannot be the case, as by inspecting the structure of M we observe that both the input guarded replication on u , and the selection on y cannot proceed on their own.
- (c) $K_{\Delta} \longrightarrow K'_{\Delta}$ and $K[M] \longrightarrow (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K'_{\Delta} \mid V \mid M)$. This reduction can be matched by $K[N]$ with a finite weak transition, as K_{Δ} occurs identically in both processes. Using subject reduction ([Theorem 3.1](#)), it can be shown that the derivatives are still in \mathcal{R} .
- (d) $V \longrightarrow V'$ and $K[M] \longrightarrow (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid V' \mid M) = M_1^*$. This case proceeds similarly, as V occurs identically in both $K[M]$ and $K[N]$.
- (e) The reduction arises from a synchronization on y between V and M . Then we have two subcases. The first one is when $V \xrightarrow{y.inr} V'$:

$$K[M] \longrightarrow (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K_{\Delta} \mid (\nu y)(V' \mid (\nu u)((!u(z).\hat{D}) \mid \hat{E}))) = M_1^*$$

This reduction can be matched by $K[N]$ via a finite weak transition:

$$K[N] \Longrightarrow (\nu \tilde{u}, \tilde{x})(K_{\Gamma} \mid K'_{\Delta} \mid (\nu y)(V'' \mid (\nu u)((!u(z).\hat{D}) \mid \hat{E}))) = N_1^*$$

where N_1^* reflects the fact that internal actions could have taken place after the synchronization on y . The typing of the process can be shown to be preserved by subject reduction ([Theorem 3.1](#)), and so $(M_1^*, N_1^*) \in \mathcal{R}$.

The second subcase is when $S \xrightarrow{y.inl} S'$; this case is similar to the first one.

2. Case $\alpha \neq \tau$: Then α corresponds to the execution of some behavior described by T , in the right-hand side typing. However, this cannot be the case since, as specified by the typings in ([C.3](#)), the behavior described by T can only be provided by \hat{E} or by \hat{F} , which are behind a selection prefix on y , both in $K[M]$ and $K[N]$. Therefore, the behavior described by T cannot be exercised until such a prefix is consumed, and we have that, necessarily, $\alpha = \tau$. Observe that once such prefixes are consumed (via internal actions) the evolution corresponding to the behavior described by T is still in \mathcal{R} , as the continuation relation $\mathcal{W}_{\vdash T}$ is in \mathcal{R} .

The analysis when $K[N] \xrightarrow{\alpha} N_1^*$ follows the same lines and is omitted. \square

C.2. Proof of [Theorem 7.2](#)

We repeat the statement of [Theorem 7.2](#) and present its full proof.

Theorem 9.1 ([Theorem 7.2](#)). *Let A, B be any type, as in [Definition 3.1](#). Then the following hold:*

- (i) $A \otimes B \simeq B \otimes A$
- (ii) $(A \oplus B) \multimap C \simeq (A \multimap C) \& (B \multimap C)$
- (iii) $!(A \& B) \simeq !A \otimes !B$.

Proof. We detail the proof of (i). We verify conditions (i)–(iv) hold for processes $P^{(x,y)}$, $Q^{(y,x)}$ defined as

$$\begin{aligned} P^{(x,y)} &= x(u).\bar{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) \\ Q^{(y,x)} &= y(w).\bar{x}(m).([y \leftrightarrow m] \mid [w \leftrightarrow x]) \end{aligned}$$

Checking (i)–(ii), i.e., $\cdot; x:A \otimes B \vdash P^{(x,y)} :: y:B \otimes A$ and $\cdot; y:B \otimes A \vdash Q^{(y,x)} :: x:A \otimes B$ is easy; for instance, the typing derivation for (i) is as follows:

$$\frac{\frac{x:B \vdash [x \leftrightarrow n] :: n:B \quad (\text{Tid}) \quad \frac{u:A \vdash [u \leftrightarrow y] :: y:A \quad (\text{Tid})}{u:A \vdash [u \leftrightarrow y] :: y:A} \quad (\text{T}\otimes\text{R})}{u:A, x:B \vdash \bar{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y:B \otimes A} \quad (\text{T}\otimes\text{L})$$

$$\frac{}{x:A \otimes B \vdash x(u).\bar{y}(n).([x \leftrightarrow n] \mid [u \leftrightarrow y]) :: y:B \otimes A}$$

Observe how the use of rule (Tid) ensures that typings hold for any A, B . We are then left to show (iii) and (iv). We sketch only the proof of (iii); the proof of (iv) is analogous. Let $M = (\nu y)(P^{(x,y)} \mid Q^{(y,z)})$, $N = [x \leftrightarrow z]$; we need to show $\cdot; x:A \otimes B \vdash M \approx N :: z:A \otimes B$. By Proposition 6.3, we have to show that for every $K \in \mathcal{K}_{\cdot; x:A \otimes B}$, we have $\vdash K[M] \approx K[N] :: z:A \otimes B$. In turn, this implies exhibiting a typed context bisimilarity \mathcal{R} containing $(K[M], K[N])$.

Letting $\mathcal{S} = \{(R_1, R_2) : K[M] \Rightarrow R_1, K[N] \Rightarrow R_2\}$, we set $\mathcal{R} = \mathcal{W}_{\vdash z:A \otimes B} \cup \mathcal{S} \cup \mathcal{S}^{-1}$. We show \mathcal{R} is a typed context bisimilarity. Pick any $K \in \mathcal{K}_{\cdot; x:A \otimes B}$. Using Definition 6.5, we can assume $K = (\nu x)(T^{(x)} \mid [\cdot])$ where $\vdash T^{(x)} :: x:A \otimes B$. By Lemma 3.2 and Theorem 5.1, there exist $l, T_1^{(x)}$ such that $T^{(x)} \xrightarrow{\bar{x}(l)} T_1^{(x)}$ in a finite transition. Clearly, $(K[M], K[N]) \in \mathcal{R}$. Now, suppose $K[N] \xrightarrow{\alpha} N_1^*$. We have to find a matching action α from $K[M]$, i.e., $K[N] \xrightarrow{\alpha} M_1^*$. $K[N]$ has only an internal action, which leads to the renaming of $T^{(x)} : K[N] \xrightarrow{\tau} T^{(z)} = N_1^*$. Using Theorem 5.1, $K[M]$ can match this action with a finite weak transition: $K[M] \Rightarrow (\nu n)(T_1^{(n)} \mid \bar{z}(m).([l \leftrightarrow m] \mid [n \leftrightarrow z])) = M_1^*$. Using Theorem 3.1, we know that $(N_1^*, M_1^*) \in \mathcal{S}^{-1}$. Now suppose $N_1^* \xrightarrow{\bar{z}(l)} T_1^{(z)}$; M_1^* can match this action with an output followed by a renaming: $M_1^* \xrightarrow{\bar{z}(m)} T_1^{(z)} \mid [l \leftrightarrow m]$. By definition of \approx (output clause), we take a process $S^{(c)}$ such that $\cdot; c:A \vdash S^{(c)} :: -!1$, and compose it with N_1^* and M_1^* . We thus obtain $N_2^* = (\nu l)(T_1^{(z)} \mid S^{(l)})$ and $M_2^* = (\nu m)(T_1^{(z)} \mid [l \leftrightarrow m] \mid S^{(m)})$; it can be easily checked that $(N_2^*, M_2^*) \in \mathcal{W}_{\vdash z:A \otimes B}$. When $K[M]$ moves first, the analysis is similar and we omit it.

The proof of (ii) uses processes

$$\cdot; x:A \oplus B \multimap C \vdash P^{(x,y)} :: (A \multimap C) \& (B \multimap C)$$

$$\cdot; y:(A \multimap C) \& (B \multimap C) \vdash Q^{(y,x)} :: x:A \oplus B \multimap C$$

defined as:

$$P^{(x,y)} = y.\text{case}(M, N) \quad \text{where } M = y(m).\bar{x}(n).(n.\text{inl}; [m \leftrightarrow n] \mid [x \leftrightarrow y])$$

$$N = y(v).\bar{x}(w).(w.\text{inr}; [v \leftrightarrow w] \mid [x \leftrightarrow y])$$

$$Q^{(y,x)} = x(m).m.\text{case}(R, S) \quad \text{where } R = y.\text{inl}; \bar{y}(n).([m \leftrightarrow n] \mid [y \leftrightarrow x])$$

$$S = y.\text{inr}; \bar{y}(w).([m \leftrightarrow w] \mid [y \leftrightarrow x])$$

The proof of (iii) uses processes

$$\cdot; x!(A \& B) \vdash P^{(x,y)} :: y:(A \otimes B)$$

$$\cdot; y:(A \otimes B) \vdash Q^{(y,x)} :: x!(A \& B)$$

defined as:

$$P^{(x,y)} = \bar{y}(n).(M \mid N) \quad \text{where } M = !n(m).\bar{x}(l).l.\text{inl}; [l \leftrightarrow m]$$

$$N = !y(h).\bar{x}(k).k.\text{inr}; [k \leftrightarrow h]$$

$$Q^{(y,x)} = y(z).!x(n).n.\text{case}(R, S) \quad \text{where } R = \bar{z}(l).[l \leftrightarrow n]$$

$$S = \bar{y}(k).[k \leftrightarrow n]. \quad \square$$

Appendix D. Supplement to Section 4: full list of commuting conversions

For convenience, below we recall the definition of \simeq_c , given in Page 262.

Definition Appendix D.1 (Proof conversions). We define \simeq_c as the least congruence on processes induced by the process equalities in Figs. D.6, D.7, D.8, and D.9.

We recall that not all permutations are sound nor are possible. In particular, for permutability of two inference rules to be sound, one of them has to be a left rule; the permutation of two right rules leads to unsound transformations. In the figures, we consider only combinations with rule (T& L₁); permutations involving (T& L₂) are easily derivable. While

$\Gamma; \Delta \vdash (\nu x)(P \mid \bar{z}(y).(Q \mid R)) \simeq_c \bar{z}(y).((\nu x)(P \mid Q) \mid R) :: z:A \otimes B$	(A-1)
$\Gamma; \Delta \vdash (\nu x)(P \mid \bar{z}(y).(Q \mid R)) \simeq_c \bar{z}(y).(Q \mid (\nu x)(P \mid R)) :: z:A \otimes B$	(A-2)
$\Gamma; \Delta, y:A \otimes B \vdash (\nu x)(P \mid y(z).Q) \simeq_c y(z).(\nu x)(P \mid Q) :: T$	(A-3)
$\Gamma; \Delta, y:A \otimes B \vdash (\nu x)(y(z).P \mid Q) \simeq_c y(z).(\nu x)(P \mid Q) :: T$	(A-4)
$\Gamma; \Delta \vdash (\nu x)(P \mid z(y).Q) \simeq_c z(y).(\nu x)(P \mid Q) :: z:A \multimap B$	(A-5)
$\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(P \mid \bar{y}(z).(Q \mid R)) \simeq_c \bar{y}(z).((\nu x)(P \mid Q) \mid R) :: T$	(A-6)
$\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(P \mid \bar{y}(z).(Q \mid R)) \simeq_c \bar{y}(z).(Q \mid (\nu x)(P \mid R)) :: T$	(A-7)
$\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(\bar{y}(z).(Q \mid P) \mid R) \simeq_c \bar{y}(z).(Q \mid (\nu x)(P \mid R)) :: T$	(A-8)
$\Gamma; \Delta \vdash (\nu x)(P \mid z.\text{case}(Q, R)) \simeq_c z.\text{case}((\nu x)(P \mid Q), (\nu x)(P \mid R)) :: z:A \& B$	(A-9)
$\Gamma; \Delta, y:A \& B \vdash (\nu x)(P \mid y.\text{inl}; Q) \simeq_c y.\text{inl}; (\nu x)(P \mid Q) :: T$	(A-10)
$\Gamma; \Delta, y:A \& B \vdash (\nu x)(P \mid y.\text{inr}; R) \simeq_c y.\text{inr}; (\nu x)(P \mid R) :: T$	(A-11)
$\Gamma; \Delta \vdash (\nu x)(P \mid z.\text{inl}; Q) \simeq_c z.\text{inl}; (\nu x)(P \mid Q) :: z:A \oplus B$	(A-12)
$\Gamma; \Delta \vdash (\nu x)(P \mid z.\text{inr}; R) \simeq_c z.\text{inr}; (\nu x)(P \mid R) :: z:A \oplus B$	(A-13)
$\Gamma; \Delta, y:A \oplus B \vdash (\nu x)(P \mid y.\text{case}(Q, R)) \simeq_c y.\text{case}((\nu x)(P \mid Q), (\nu x)(P \mid R)) :: T$	(A-14)
$\Gamma, u:A; \Delta \vdash (\nu x)(P \mid \bar{u}(y).Q) \simeq_c \bar{u}(y).(\nu x)(P \mid Q) :: T$	(A-15)
$\Gamma; \Delta, y:A \otimes B \vdash (\nu x)(y(z).P \mid R) \simeq_c y(z).(\nu x)(P \mid R) :: T$	(B-1)
$\Gamma; \Delta, y:A \multimap B \vdash (\nu x)(\bar{y}(z).(P \mid Q) \mid R) \simeq_c \bar{y}(z).(P \mid (\nu x)(Q \mid R)) :: T$	(B-2)
$\Gamma; \Delta, y:A \& B \vdash (\nu x)(y.\text{inl}; P \mid R) \simeq_c y.\text{inl}; (\nu x)(P \mid R) :: T$	(B-3)
$\Gamma; \Delta, y:A \& B \vdash (\nu x)(y.\text{inr}; P \mid R) \simeq_c y.\text{inr}; (\nu x)(P \mid R) :: T$	(B-4)
$\Gamma; \Delta, y:A \oplus B \vdash (\nu x)(y.\text{case}(P, Q) \mid R) \simeq_c y.\text{case}((\nu x)(P \mid R), (\nu x)(Q \mid R)) :: T$	(B-5)
$\Gamma; \Delta \vdash (\nu x)(P \mid \{y/u\} \mid Q) \simeq_c (\nu x)(P \mid Q) \{y/u\} :: T$	(B-6)
$\Gamma; \Delta \vdash (\nu x)(P \mid Q \mid \{y/u\}) \simeq_c (\nu x)(P \mid Q) \{y/u\} :: T$	(B-7)
$\Gamma, u:A; \Delta \vdash (\nu x)(\bar{u}(y).P \mid R) \simeq_c \bar{u}(y).(\nu x)(P \mid R) :: T$	(B-8)
$\Gamma, u:A; \Delta \vdash (\nu x)(P \mid \bar{u}(y).R) \simeq_c \bar{u}(y).(\nu x)(P \mid R) :: T$	(B-9)

Fig. D.6. Process equalities induced by proof conversions: classes (A) and (B).

$\Gamma; \cdot \vdash (\nu u)((!u(y).P) \mid \mathbf{0}) \simeq_c \mathbf{0} :: \cdot : \mathbf{1}$	(C-1)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid \bar{x}(z).(Q \mid R)) \simeq_c \bar{x}(z).((\nu u)((!u(y).P) \mid Q) \mid (\nu u)((!u(y).P) \mid R)) :: x:A \otimes B$	(C-2)
$\Gamma; \Delta, y:A \otimes B \vdash (\nu u)((!u(y).P) \mid y(z).Q) \simeq_c y(z).(\nu u)((!u(y).P) \mid Q) :: T$	(C-3)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid z(y).Q) \simeq_c z(y).(\nu u)((!u(y).P) \mid Q) :: z:A \multimap B$	(C-4)
$\Gamma; \Delta, y:A \multimap B \vdash (\nu u)((!u(w).P) \mid \bar{y}(z).(Q \mid R)) \simeq_c \bar{y}(z).(((\nu u)((!u(w).P) \mid Q) \mid (\nu u)((!u(w).P) \mid R))) :: T$	(C-5)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid z.\text{case}(Q, R)) \simeq_c z.\text{case}((\nu u)((!u(y).P) \mid Q), (\nu u)((!u(y).P) \mid R)) :: z:A \& B$	(C-6)
$\Gamma; \Delta, y:A \& B \vdash (\nu u)((!u(z).P) \mid y.\text{inl}; Q) \simeq_c y.\text{inl}; (\nu u)((!u(z).P) \mid Q) :: T$	(C-7)
$\Gamma; \Delta, y:A \& B \vdash (\nu u)((!u(z).P) \mid y.\text{inr}; R) \simeq_c y.\text{inr}; (\nu u)((!u(z).P) \mid R) :: T$	(C-8)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid z.\text{inl}; Q) \simeq_c z.\text{inl}; (\nu u)((!u(y).P) \mid Q) :: z:A \oplus B$	(C-9)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid z.\text{inr}; R) \simeq_c z.\text{inr}; (\nu u)((!u(y).P) \mid R) :: z:A \oplus B$	(C-10)
$\Gamma; \Delta, y:A \oplus B \vdash (\nu u)((!u(z).P) \mid y.\text{case}(Q, R)) \simeq_c y.\text{case}((\nu u)((!u(z).P) \mid Q), (\nu u)((!u(z).P) \mid R)) :: T$	(C-11)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid !x(z).Q) \simeq_c !x(z).(\nu u)((!u(y).P) \mid Q) :: x:!A$	(C-12)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid Q \mid \{y/v\}) \simeq_c (\nu u)((!u(y).P) \mid Q) \{y/v\} :: T$	(C-13)
$\Gamma; \Delta \vdash (\nu u)((!u(y).P) \mid \bar{v}(y).Q) \simeq_c \bar{v}(y).(\nu u)((!u(y).P) \mid Q) :: T$	(C-14)

Fig. D.7. Process equalities induced by proof conversions: class (C).

there is no rule that can permute with (T1R), rule (T1L) can permute with all rules without changing the process structure. The situation is similar for (T!R) and (T!L): the former is incompatible for permutation with all rules, while the latter can permute with all rules, excepting (T!R). The effect of (T!L) in processes is a substitution; equated processes only differ in the scope of such a substitution.

$\Gamma; \Delta, x:A \otimes B, z:C \otimes D \vdash x(y).z(w).P \simeq_c z(w).x(y).P :: T$	(D-1)
$\Gamma; \Delta, z:D \multimap C, x:A \multimap B \vdash \bar{z}(w).(R \mid \bar{x}(y).(P \mid Q)) \simeq_c \bar{x}(y).(P \mid \bar{z}(w).(R \mid Q)) :: T$	(D-2)
$\Gamma; \Delta, z:D \multimap C, x:A \multimap B \vdash \bar{z}(w).(R \mid \bar{x}(y).(P \mid Q)) \simeq_c \bar{x}(y).(\bar{z}(w).(R \mid P) \mid Q) :: T$	(D-3)
$\Gamma; \Delta, w:C \multimap D, x:A \otimes B \vdash \bar{w}(z).(Q \mid x(y).P) \simeq_c x(y).\bar{w}(z).(Q \mid P) :: T$	(D-4)
$\Gamma; \Delta, w:C \multimap D, x:A \otimes B \vdash \bar{w}(z).(x(y).P \mid Q) \simeq_c x(y).\bar{w}(z).(P \mid Q) :: T$	(D-5)
$\Gamma, u:A, v:C; \Delta \vdash \bar{u}(y).\bar{v}(x).P \simeq_c \bar{v}(x).\bar{u}(y).P :: T$	(D-6)
$\Gamma, u:C; \Delta, x:A \multimap B \vdash \bar{u}(z).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(\bar{u}(z).P \mid Q) :: T$	(D-7)
$\Gamma, u:C; \Delta, x:A \multimap B \vdash \bar{u}(z).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(P \mid \bar{u}(z).Q) :: T$	(D-8)
$\Gamma, u:A; \Delta, z:C \otimes D \vdash \bar{u}(y).z(w).P \simeq_c z(w).\bar{u}(y).P :: T$	(D-9)
$\Gamma; \Delta, x:A \oplus B, y:C \oplus D \vdash y.\text{case}(x.\text{case}(P_1, Q_1), x.\text{case}(P_2, Q_2)) \simeq_c x.\text{case}(y.\text{case}(P_1, P_2), y.\text{case}(Q_1, Q_2)) :: T$	(D-10)
$\Gamma, u:C; \Delta, x:A \oplus B \vdash \bar{u}(z).x.\text{case}(P, Q) \simeq_c x.\text{case}(\bar{u}(z).P, \bar{u}(z).Q) :: T$	(D-11)
$\Gamma; \Delta, w:A \multimap E, z:C \oplus D \vdash z.\text{case}(\bar{w}(y).(P \mid R_1), \bar{w}(y).(P \mid R_2)) \simeq_c \bar{w}(y).(P \mid z.\text{case}(R_1, R_2)) :: T$	(D-12)
$\Gamma; \Delta, z:C \oplus D, x:A \otimes B \vdash z.\text{case}(x(y).P, x(y).Q) \simeq_c x(y).z.\text{case}(P, Q) :: T$	(D-13)
$\Gamma; \Delta, x:A \& B, y:C \& D \vdash x.\text{inl}; y.\text{inl}; P \simeq_c y.\text{inl}; x.\text{inl}; P :: T$	(D-14)
$\Gamma; \Delta, x:A \oplus B, y:C \& D \vdash x.\text{case}(y.\text{inl}; P, y.\text{inl}; Q) \simeq_c y.\text{inl}; x.\text{case}(P, Q) :: T$	(D-15)
$\Gamma, u:C; \Delta, z:A \& B \vdash z.\text{inl}; \bar{u}(y).P \simeq_c \bar{u}(y).z.\text{inl}; P :: T$	(D-16)
$\Gamma; \Delta, z:C \& D, x:A \multimap B \vdash z.\text{inl}; \bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(z.\text{inl}; P \mid Q) :: T$	(D-17)
$\Gamma; \Delta, z:C \& D, x:A \multimap B \vdash z.\text{inl}; \bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(P \mid z.\text{inl}; Q) :: T$	(D-18)
$\Gamma; \Delta, z:C \& D, x:A \otimes B \vdash z.\text{inl}; x(y).P \simeq_c x(y).z.\text{inl}; P :: T$	(D-19)

Fig. D.8. Process equalities induced by proof conversions: class (D).

$\Gamma; \Delta, z:C \& D \vdash z.\text{inl}; \bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(P \mid z.\text{inl}; Q) :: x:A \otimes B$	(E-1)
$\Gamma; \Delta, z:C \& D \vdash z.\text{inl}; \bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(z.\text{inl}; P \mid Q) :: x:A \otimes B$	(E-2)
$\Gamma; \Delta, z:D \oplus E \vdash z.\text{case}(\bar{x}(y).(P_1 \mid Q), \bar{x}(y).(P_2 \mid Q)) \simeq_c \bar{x}(y).(Q \mid z.\text{case}(P_1, P_2)) :: x:A \otimes B$	(E-3)
$\Gamma; \Delta, z:D \oplus E \vdash z.\text{case}(\bar{x}(y).(Q \mid P_1), \bar{x}(y).(Q \mid P_2)) \simeq_c \bar{x}(y).(z.\text{case}(P_1, P_2) \mid Q) :: x:A \otimes B$	(E-4)
$\Gamma, u:C; \Delta \vdash \bar{u}(w).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(\bar{u}(w).P \mid Q) :: x:A \otimes B$	(E-5)
$\Gamma, u:C; \Delta \vdash \bar{u}(w).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(P \mid \bar{u}(w).Q) :: x:A \otimes B$	(E-6)
$\Gamma; \Delta, w:C \multimap D \vdash \bar{w}(z).(R \mid \bar{x}(y).(P \mid Q)) \simeq_c \bar{x}(y).(P \mid \bar{w}(z).(R \mid Q)) :: x:A \otimes B$	(E-7)
$\Gamma; \Delta, x:C \multimap D \vdash \bar{z}(y).(\bar{x}(w).(P \mid Q) \mid R) \simeq_c \bar{x}(w).(P \mid \bar{z}(y).(R \mid Q)) :: z:A \otimes B$	(E-8)
$\Gamma; \Delta, z:C \otimes D \vdash z(w).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(z(w).P \mid Q) :: x:A \otimes B$	(E-9)
$\Gamma; \Delta, z:C \otimes D \vdash z(w).\bar{x}(y).(P \mid Q) \simeq_c \bar{x}(y).(P \mid z(w).Q) :: x:A \otimes B$	(E-10)
$\Gamma; \Delta, z:C \& D \vdash z.\text{inl}; x(y).P \simeq_c x(y).z.\text{inl}; P :: x:A \multimap B$	(E-11)
$\Gamma; \Delta, z:C \oplus D \vdash x(y).z.\text{case}(P, Q) \simeq_c z.\text{case}(x(y).P, x(y).Q) :: x:A \multimap B$	(E-12)
$\Gamma, u:C; \Delta \vdash \bar{u}(w).x(y).P \simeq_c x(y).\bar{u}(w).P :: x:A \multimap B$	(E-13)
$\Gamma; \Delta, w:C \multimap D \vdash \bar{w}(z).(R \mid x(y).P) \simeq_c x(y).\bar{w}(z).(R \mid P) :: x:A \multimap B$	(E-14)
$\Gamma; \Delta, z:C \otimes D \vdash x(y).z(w).P \simeq_c z(w).x(y).P :: x:A \multimap B$	(E-15)
$\Gamma; \Delta, y:C \& D \vdash y.\text{inl}; x.\text{case}(P, Q) \simeq_c x.\text{case}(y.\text{inl}; P, y.\text{inl}; Q) :: x:A \& B$	(E-16)
$\Gamma; \Delta, y:C \oplus D \vdash x.\text{case}(y.\text{case}(P_1, Q_1), y.\text{case}(P_2, Q_2)) \simeq_c y.\text{case}(x.\text{case}(P_1, P_2), x.\text{case}(Q_1, Q_2)) :: x:A \& B$	(E-17)
$\Gamma; u:A; \Delta \vdash x.\text{case}(\bar{u}(y).P, \bar{u}(y).Q) \simeq_c \bar{u}(y).x.\text{case}(P, Q) :: x:A \& B$	(E-18)
$\Gamma; \Delta, z:C \multimap D \vdash \bar{z}(y).(R \mid x.\text{case}(P, Q)) \simeq_c x.\text{case}(\bar{z}(y).(R \mid P), \bar{z}(y).(R \mid Q)) :: x:A \& B$	(E-19)
$\Gamma; \Delta, x:A \otimes B \vdash z.\text{case}(x(y).P, x(y).Q) \simeq_c x(y).z.\text{case}(P, Q) :: z:C \& D$	(E-20)
$\Gamma; \Delta, y:C \& D \vdash y.\text{inl}; x.\text{inl}; P \simeq_c x.\text{inl}; y.\text{inl}; P :: x:A \oplus B$	(E-21)
$\Gamma; \Delta, y:A \oplus B \vdash x.\text{inl}; y.\text{case}(P, Q) \simeq_c y.\text{case}(x.\text{inl}; P, x.\text{inl}; Q) :: x:A \oplus B$	(E-22)
$\Gamma; u:A; \Delta \vdash x.\text{inl}; \bar{u}(y).P \simeq_c \bar{u}(y).x.\text{inl}; P :: x:A \oplus B$	(E-23)
$\Gamma; \Delta, z:D \multimap C \vdash \bar{z}(y).(Q \mid x.\text{inl}; P) \simeq_c x.\text{inl}; \bar{z}(y).(Q \mid P) :: x:A \oplus B$	(E-24)
$\Gamma; \Delta, x:A \otimes B \vdash x(y).z.\text{inl}; P \simeq_c z.\text{inl}; x(y).P :: z:C \oplus D$	(E-25)
$\Gamma; \Delta, x:C \vdash \bar{z}(y).(P\{x/u\} \mid Q) \simeq_c (\bar{z}(y).(P \mid Q))\{x/u\} :: z:A \otimes B$	(E-26)
$\Gamma; \Delta, x:C \vdash \bar{z}(y).(P \mid Q\{x/u\}) \simeq_c (\bar{z}(y).(P \mid Q))\{x/u\} :: z:A \otimes B$	(E-27)

Fig. D.9. Process equalities induced by proof conversions: class (E).

References

- [1] S. Abramsky, Computational interpretations of linear logic, *Theor. Comput. Sci.* 111 (April 1993) 3–57.
- [2] A. Barber, Dual intuitionistic linear logic, Technical report, LFCS-96-347, Univ. of Edinburgh, 1996.
- [3] M. Berger, K. Honda, N. Yoshida, Genericity and the pi-calculus, in: *FoSSaCS*, in: LNCS, vol. 2620, Springer, 2003, pp. 103–119.
- [4] M. Berger, K. Honda, N. Yoshida, Genericity and the pi-calculus, *Acta Inform.* 42 (2–3) (2005) 83–141.
- [5] M. Boreale, On the expressiveness of internal mobility in name-passing calculi, *Theor. Comput. Sci.* 195 (March 1998) 205–226.
- [6] M. Boreale, D. Sangiorgi, Bisimulation in name-passing calculi without matching, in: *LICS*, 1998, pp. 165–175.
- [7] G. Boudol, Typing termination in a higher-order concurrent imperative language, *Inf. Comput.* 208 (6) (2010) 716–736.
- [8] L. Caires, Logical semantics of types for concurrency, in: *CALCO* 2007, in: LNCS, vol. 4624, Springer, 2007, pp. 16–35.
- [9] L. Caires, J.A. Pérez, F. Pfenning, B. Toninho, Relational parametricity for polymorphic session types, Technical report, CMU-CS-12-108, Carnegie Mellon University, April 2012.
- [10] L. Caires, J.A. Pérez, F. Pfenning, B. Toninho, Behavioral polymorphism and parametricity in session-based communication, in: M. Felleisen, P. Gardner (Eds.), *ESOP*, in: LNCS, vol. 7792, Springer, 2013, pp. 330–349.
- [11] L. Caires, F. Pfenning, Session types as intuitionistic linear propositions, in: *CONCUR'2010*, in: LNCS, vol. 6269, Springer, 2010, pp. 222–236.
- [12] L. Caires, F. Pfenning, B. Toninho, Linear logic propositions as session types, *Math. Struct. Comput. Sci.* (2014), in press, <http://www.cs.cmu.edu/~btoninho/mscs12.pdf>.
- [13] B.-Y.E. Chang, K. Chaudhuri, F. Pfenning, A judgmental analysis of linear logic, Technical report, CMU-CS-03-131R, Carnegie Mellon University, 2003.
- [14] U. Dal Lago, P. Di Gianberardino, Soft session types, in: *Proc. of 18th Workshop on Expressiveness in Concurrency, EXPRESS'11*, in: *EPTCS*, vol. 64, 2011, pp. 59–73.
- [15] R. Demangeon, D. Hirschhoff, D. Sangiorgi, Mobile processes and termination, in: *Semantics and Algebraic Specification*, in: LNCS, vol. 5700, Springer, 2009, pp. 250–273.
- [16] Y. Deng, D. Sangiorgi, Towards an algebraic theory of typed mobile processes, *Theor. Comput. Sci.* 350 (2–3) (2006) 188–212.
- [17] M. Dezani-Ciancaglini, U. de'Liguoro, Sessions and session types: an overview, in: *WS-FM* 2009, in: LNCS, vol. 6194, Springer, 2010, pp. 1–28.
- [18] R. Di Cosmo, A short survey of isomorphisms of types, *Math. Struct. Comput. Sci.* 15 (5) (2005) 825–838.
- [19] J.-Y. Girard, Y. Lafont, Linear logic and lazy computation, in: *TAPSOFT'87*, in: LNCS, vol. 250, Springer, 1987, pp. 52–66.
- [20] J.F. Groote, M.P.A. Sellink, Confluence for process verification, *Theor. Comput. Sci.* 170 (1–2) (1996) 47–81.
- [21] M. Hennessy, J. Rathke, Typed behavioural equivalences for processes in the presence of subtyping, *Math. Struct. Comput. Sci.* 14 (5) (2004) 651–684.
- [22] K. Honda, Types for dynamic interaction, in: *CONCUR*, in: LNCS, vol. 715, Springer, 1993, pp. 509–523.
- [23] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: *ESOP'98*, in: LNCS, vol. 1381, Springer, 1998, pp. 122–138.
- [24] W.A. Howard, The formulae-as-types notion of construction, Unpublished note. An annotated version appeared in: To H.B. Curry: *Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980, pp. 479–490.
- [25] R. Hu, N. Yoshida, K. Honda, Session-based distributed programming in java, in: *Proc. of ECOOP*, in: LNCS, vol. 5142, Springer, 2008, pp. 516–541.
- [26] N. Kobayashi, B.C. Pierce, D.N. Turner, Linearity and the pi-calculus, in: *POPL*, 1996, pp. 358–371.
- [27] D. Kouzapas, N. Yoshida, R. Hu, K. Honda, On asynchronous session semantics, in: *Proc. of FMOODS-FORTE'2011*, in: LNCS, vol. 6722, Springer, 2011, pp. 228–243.
- [28] Y. Lafont, Soft linear logic and polynomial time, *Theor. Comput. Sci.* 318 (1–2) (2004) 163–180.
- [29] K. Mazurak, S. Zdancewicz, Lollipop: to concurrency from classical linear logic via Curry–Howard and control, in: *ICFP'10*, ACM, 2010, pp. 39–50.
- [30] R. Milner, *Communication and Concurrency*, Prentice Hall, 1995.
- [31] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, part I/II, *Inf. Comput.* 100 (1) (1992) 1–77.
- [32] U. Nestmann, On determinacy and nondeterminacy in concurrent programming, PhD thesis, Technische Fakultät, Universität Erlangen, 1996.
- [33] N. Ng, N. Yoshida, O. Pernet, R. Hu, Y. Kiyftis, Safe parallel programming with session Java, in: *Proc. of COORDINATION*, in: LNCS, vol. 6721, Springer, 2011, pp. 110–126.
- [34] J.A. Pérez, L. Caires, F. Pfenning, B. Toninho, Linear logical relations for session-based concurrency, in: *ESOP*, in: LNCS, vol. 7211, Springer, 2012, pp. 539–558.
- [35] F. Pfenning, L. Caires, B. Toninho, Proof-carrying code in a session-typed process calculus, in: *Proc. of CPP'11*, in: LNCS, vol. 7086, Springer, 2011, pp. 21–36.
- [36] A. Philippou, D. Walker, On confluence in the pi-calculus, in: *ICALP*, in: LNCS, vol. 1256, Springer, 1997, pp. 314–324.
- [37] B.C. Pierce, D. Sangiorgi, Typing and subtyping for mobile processes, *Math. Struct. Comput. Sci.* 6 (5) (1996) 409–453.
- [38] B.C. Pierce, D. Sangiorgi, Behavioral equivalence in the polymorphic pi-calculus, *J. ACM* 47 (3) (2000) 531–584.
- [39] R. Pucella, J.A. Tov, Haskell session types with (almost) no class, in: *Proc. of ACM SIGPLAN Symposium on Haskell*, ACM, 2008, pp. 25–36.
- [40] D. Sangiorgi, pi-Calculus, internal mobility, and agent-passing calculi, *Theor. Comput. Sci.* 167 (1–2) (1996) 235–274.
- [41] D. Sangiorgi, Termination of processes, *Math. Struct. Comput. Sci.* 16 (1) (2006) 1–39.
- [42] D. Sangiorgi, D. Walker, *The π -Calculus: A Theory of Mobile Processes*, Cambridge University Press, New York, NY, USA, 2001.
- [43] R. Statman, Logical relations and the typed lambda-calculus, *Inf. Control* 65 (2/3) (1985) 85–97.
- [44] W.W. Tait, Intensional interpretations of functionals of finite type I, *J. Symb. Log.* 32 (1967) 198–212.
- [45] B. Toninho, L. Caires, F. Pfenning, Dependent session types via intuitionistic linear type theory, in: *Proc. of PPDP'11*, ACM, New York, NY, USA, 2011, pp. 161–172.
- [46] P. Wadler, Propositions as sessions, in: P. Thiemann, R.B. Findler (Eds.), *ICFP*, ACM, 2012, pp. 273–286.
- [47] N. Yoshida, M. Berger, K. Honda, Strong normalisation in the pi-calculus, *Inf. Comput.* 191 (2) (2004) 145–202.
- [48] N. Yoshida, K. Honda, M. Berger, Linearity and bisimulation, *J. Log. Algebr. Program.* 72 (2) (2007) 207–238.