Depending on Session-Typed Processes

Bernardo Toninho and Nobuko Yoshida

Imperial College London, United Kingdom

Abstract. This work proposes a dependent type theory that combines functions and session-typed processes (with value dependencies) through a contextual monad, internalising typed processes in a dependently-typed λ-calculus. The proposed framework, by allowing session processes to depend on functions and vice-versa, enables us to specify and statically verify protocols where the choice of the next communication action can depend on specific values of received data. Moreover, the type theoretic nature of the framework endows us with the ability to internally describe and prove predicates on process behaviours. Our main results are type soundness of the framework, and a faithful embedding of the functional layer of the calculus within the session-typed layer, showcasing the expressiveness of dependent session types.

1 Introduction

Session types [26,14] are a typing discipline for communication protocols, whose simplicity provides an extensible framework that allows for integration with a variety of functional type features. One useful instance arising from the proof theoretic exploration of logical quantification is *value dependent session types* [27]. In this work, one can express properties of exchanged data in protocol specifications separately from communication, but *cannot* describe protocols where communication actions depend on the actual exchanged data (e.g. [17, § 2]). Moreover, it does not allow functions or values to depend on protocols (i.e. sessions) or communication, thus preventing reasoning about dependent process behaviours, exploring the proofs-as-programs paradigm of dependent type theory, e.g. [18,8].

Our work addresses the limitations of existing formulations of session types by proposing a type theory that integrates dependent functions *and* session types using a *contextual monad.* This monad internalises a session-typed calculus within a dependently-typed λ-calculus. By allowing session types to depend on λ-terms *and* λ-terms to depend on typed processes (using the monad), we are able to achieve heightened degrees of expressiveness. Exploiting the former direction, we enable writing actual data-dependent communication protocols. Exploiting the latter, we can define and *prove* properties of linearly-typed objects (i.e. processes) within our intuitionistic theory.

To informally demonstrate how our type theory goes beyond the state of the art in order to represent data-dependent protocols, consider the following session type (we write $\tau \wedge A$ for $\exists x{:}\tau.A$ where x does not occur in A and similarly $\tau \supset A$ for $\forall x{:}\tau.A$ when x is not free in A), $T \triangleq \mathsf{Bool} \supset \oplus\{\mathtt{t} : \mathsf{Nat} \wedge \mathbf{1}, \mathtt{f} : \mathsf{Bool} \wedge \mathbf{1}\}$, representable in existing session typing systems. The type T denotes a protocol which first, inputs a boolean and then either emits the label `t`, which will be followed by an output of a natural number; or emits the label `f` and a boolean. The intended protocol described by T is to take the `t` branch if the received value is `t` and the `f` branch otherwise, which we can implement as Q with channel z typed by T as follows:

$$Q \triangleq z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z.\mathtt{t}; z\langle 23\rangle.\mathbf{0},\ \mathsf{false} \Rightarrow z.\mathtt{f}; z\langle \mathsf{true}\rangle.\mathbf{0})$$

where $z(x).P$ denotes an input process, $z.\mathtt{t}$ is a process which selects label `t` and $z\langle 23\rangle.P$ is an output on z. However, since the specification is imprecise, process $z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{false} \Rightarrow z.\mathtt{t}; z\langle 23\rangle.\mathbf{0},\ \mathsf{true} \Rightarrow z.\mathtt{f}; z\langle \mathsf{true}\rangle.\mathbf{0})$ is also a type-correct implementation of T that does not adhere to the intended protocol. Using our dependent type system, we can narrow the specification to guarantee that the desired protocol is precisely enforced. Consider the following definition of a session-type level conditional where we assume inductive definition and dependent pattern matching mechanisms (stype denotes the *kind* of session types):

$$\begin{array}{l}\mathtt{if} :: \mathsf{Bool} \to \mathsf{stype} \to \mathsf{stype} \to \mathsf{stype} \\ \mathtt{if}\ \mathsf{true}\, A\, B \ = \ A \qquad \mathtt{if}\ \mathsf{false}\, A\, B \ = \ B\end{array}$$

The type-level function above case analyses the boolean and produces its first session type argument if the value is true and the second otherwise. We may now specify a session type that faithfully implements the protocol:

$$T' \triangleq \forall x{:}\mathsf{Bool}.\mathtt{if}\ x\ (\mathsf{Nat} \wedge \mathbf{1})\ (\mathsf{Bool} \wedge \mathbf{1})$$

A process R implementing such a type on channel z is given below:

$$R \triangleq z(x).\mathsf{case}\ x\ \mathsf{of}\ (\mathsf{true} \Rightarrow z\langle 23\rangle.\mathbf{0},\ \mathsf{false} \Rightarrow z\langle \mathsf{true}\rangle.\mathbf{0})$$

Note that if we flip the two branches of the case analysis in R, the session is no longer typable with T', ensuring that the protocol is implemented faithfully.

The example above illustrates a simple yet useful data-dependent protocol. When we further extend our dependent types with a *process* monad [31], where $\{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$ is a functional term denoting a process that may be *spawned* by other processes by instantiating the names in $\overline{u_j}$ and $\overline{d_i}$, we can provide more powerful reasoning on processes, enabling refined specifications through the use of type indices (i.e. type families) and an ability to internally specify and verify predicates on process behaviours. We also show that *all* functional types and terms can be faithfully embedded in the process layer using the dependently-typed sessions and process monads.

Contributions. § 2 introduces our dependent type theory, augmenting the example above by showing how we can reason about process behaviour using type families and dependently-typed functions (§ 2.3). We then establish the soundness of the theory (§ 2.4). § 3 develops a faithful embedding of the dependent function space in the process layer (Theorem 3.4). § 4 concludes with related work. This article is a long version of [34] containing omitted definitions, proofs and additional examples.

$$
\begin{array}{lll}
\text{Kinds} & K, K' & ::= \mathsf{type} \mid \mathsf{stype} \mid \mathit{\Pi} x{:}\tau.K \mid \mathit{\Pi} t{:}K.K' \\
\text{Functional} & \tau, \sigma & ::= \mathit{\Pi} x{:}\tau.\sigma \mid \lambda x{:}\tau.\sigma \mid \tau\ M \mid \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\} \mid \lambda t :: K.\tau \mid \tau\,\sigma \\
\text{Sessions} & A, B & ::= !A \mid A \multimap B \mid A \otimes B \mid \forall x{:}\tau.A \mid \exists x{:}\tau.A \mid \mathbf{1} \\
 & & \quad \mid \ \&\{\overline{l_i : A_i}\} \mid \oplus\{\overline{l_i : A_i}\} \mid \lambda x{:}\tau.A \mid A\ M \mid \lambda t{::}K.A \mid A\,B \\
\text{Terms} & M, N & ::= \lambda x{:}\tau.M \mid \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} \mid M\,N \mid x \\
\text{Processes} & P, Q & ::= \overline{c}\langle d\rangle.P \mid (\boldsymbol{\nu} c)P \mid c(x).P \mid c\langle M\rangle.P \mid !c(x).P \\
 & & \quad \mid \ c.\mathsf{case}\{\overline{l_i \Rightarrow P_i}\} \mid c.l; P \mid [c \leftrightarrow d] \mid \mathbf{0} \mid c \leftarrow M \leftarrow \overline{u_j}; \overline{d_i}; Q
\end{array}
$$

Fig. 1. Syntax of Kinds, Types, Terms and Processes

2 A Dependent Type Theory of Processes

This section introduces our dependent type theory combining session-typed processes and functions. The theory is a generalisation of the line of work relating linear logic and session types [4,27,31], considering type-level functions and dependent kinds in an intensional type theory with full *mutual* dependencies between functions and processes. This generalisation enables us to express more sophisticated session types (such as those of § 1) and also to define and *prove* properties of processes expressed as type families with proofs as their inhabitants. We focus on the new rules and judgements, pointing the interested reader to [27,5,28] for additional details on the base theory.

2.1 Syntax

The calculus is stratified into two mutually dependent layers of processes and terms, which we often refer to as the *process* and *functional* layers, respectively. The syntax of the theory is given in Fig. 1 (we use x, y for variables ranging over terms and t for variables ranging over types).

Types and Kinds. The process layer is able to refer to terms of the functional layer via appropriate (dependently-typed) communication actions and through a *spawn* construct, allowing for processes encapsulated as functional values to be executed. Dually, the functional layer can refer to the process layer via a *contextual* monad [31] that internalises (open) typed processes as opaque functional values. This mutual dependency is also explicit in the type structure on several axes: process channel usages are typed by a language of session types, which specifies the communication protocols implemented on the used channels, extended with two dependent communication operations $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$, where τ is a functional type and A is a session type in which x may occur. Moreover, we also extend the language of session types with type-level λ-abstraction over terms $\lambda x{:}\tau.A$ and session types $\lambda t{::}K.A$ (with the corresponding elimination forms $A\,M$ and $A\,B$). As we show in § 1, the combination of these features allows for a new degree of expressiveness, enabling us to construct session types whose structure depends on previously communicated values.

The remaining session constructs are standard, following [5]: $!A$ denotes a *shared* session of type A that may be used an arbitrary (finite) number of times;

$A \multimap B$ represents a session offering to input a session of type A to then offer the session behaviour B; $A \otimes B$ is the dual operator, denoting a session that outputs A and proceeds as B; $\oplus\{\overline{l_i : A_i}\}$ and $\&\{\overline{l_i : A_i}\}$ represent internal and external labelled choice, respectively; $\mathbf{1}$ denotes the terminated session.

The functional layer is a λ-calculus with dependent functions $\Pi x{:}\tau.\sigma$, type-level λ-abstractions over terms and types (and respective type-level applications) and a *contextual monadic* type $\{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}$, denoting a (quoted) process offering session $c{:}A$ by using the *linear* sessions $\overline{d_i{:}A_i}$ and *shared* sessions $\overline{u_j{:}B_j}$ [31]. We often write $\{A\}$ for $\{\cdot;\cdot \vdash c{:}A\}$. The kinding system for our theory contains two base kinds type and stype of functional and session types, respectively. Type-level λ-abstractions require dependent kinds $\Pi x{:}\tau.K$ and $\Pi t{::}K.K'$, respectively. We note that the functional connectives form a standard dependent type theory [11,23].

Terms and Processes. Terms include the standard λ-abstractions $\lambda x{:}\tau.M$, applications $M\,N$ and variables x. In order to internalise processes within the functional layer we make use of a monadic process wrapper, written $\{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$. In such a construct, the channels c, $\overline{u_j}$ and $\overline{d_i}$ are bound in P, where c is the session channel being offered and $\overline{u_j}$ and $\overline{d_i}$ are the session channels (linear and shared, respectively) being used. We write $\{c \leftarrow P \leftarrow \epsilon\}$ when P does not use any ambient channels, which we abbreviate to $\{P\}$.

The syntax of processes follows that of [5] extended with the monadic elimination form $c \leftarrow M \leftarrow \overline{u_j}; \overline{d_i}; Q$. Such a process construct denotes a term M that is to be evaluated to a monadic value of the form $\{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$ which will then be executed in parallel with Q, sharing with it a session channel c and using the provided channels $\overline{u_j}$ and $\overline{d_i}$. We write $c \leftarrow M \leftarrow \epsilon; Q$ when no channels are provided for the execution of M and often abbreviate this to $c \leftarrow M; Q$. The process $\overline{c}\langle d\rangle.P$ denotes the output of the *fresh* channel d along channel c with continuation P, which binds d; $(\boldsymbol{\nu} c)P$ denotes channel hiding, restricting the scope of c to P; $c(x).P$ denotes an input along c, bound to x in P; $c\langle M\rangle.P$ denotes the output of term M along c with continuation P; $!c(x).P$ denotes a replicated input which spawns copies of P; the construct $c.\mathsf{case}\{\overline{l_i \Rightarrow P_i}\}$ codifies a process that waits to receive some label l_j along c, with continuation P_j; dually, $c.l; P$ denotes a process that emits a label l along c and continues as P; $[c \leftrightarrow d]$ denotes a forwarder between c and d, which is operationally implemented as renaming; $P \mid Q$ denotes parallel composition and $\mathbf{0}$ the null process.

2.2 A Dependent Typing System

We now introduce our typing system, defined by a series of mutually inductive judgements, given in Fig. 2. We use Ψ to stand for a typing context for dependent λ-terms (i.e. assumptions of the form $x{:}\tau$ or $t :: K$, not subject to exchange), Γ for a typing context for *shared* sessions of the form $u{:}A$ (implicitly subject to weakening and contraction) and Δ for a linear context of sessions $x{:}A$. The context well-formedness judgments $\Psi \vdash$ and $\Psi; \Delta \vdash$ require that types and kinds (resp. session types) in Ψ (resp. Δ) are well-formed. The judgments $\Psi \vdash K$,

$\Psi \vdash$	Context Ψ is well-formed.
$\Psi; \Delta \vdash$	Context Δ is well-formed, under assumptions in Ψ.
$\Psi \vdash K$	K is a kind in context Ψ.
$\Psi \vdash \tau :: K$	τ is a (functional) type of kind K in context Ψ.
$\Psi \vdash A :: K$	A is a session type of kind K in context Ψ.
$\Psi \vdash M : \tau$	M has type τ in context Ψ.
$\Psi; \Gamma; \Delta \vdash P :: z{:}A$	P offers session $z{:}A$ when composed with processes offering sessions specified in Γ and Δ in context Ψ.
$\Psi \vdash K_1 = K_2$	Kinds K_1 and K_2 are equal.
$\Psi \vdash \tau = \sigma :: K$	Types τ and σ are equal of kind K.
$\Psi \vdash A = B :: K$	Session types A and B are equal of kind K.
$\Psi \vdash M = N : \tau$	Terms M and N are equal of type τ.
$\Psi \vdash \Delta = \Delta' :: \mathsf{stype}$	Contexts Δ and Δ' are equal, under the assumptions in Ψ.
$\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$	Processes P and Q are equal with typing $z{:}A$.

Fig. 2. Typing Judgements

$\Psi \vdash \tau :: K$ and $\Psi \vdash A :: K$ codify well-formedness of kinds, functional and session types (with kind K), respectively. Their rules are standard.

Typing. An excerpt of the typing rules for terms and processes is given in Fig. 3 and 4, respectively, noting that typing enforces types to be of base kind type (respectively stype). The rules for dependent functions are standard, including the type conversion rule which internalises definitional equality of types. We highlight the introduction rule for the monadic construct, which requires the appropriate session types to be well-formed and the process P to offer $c{:}A$ when provided with the appropriate session contexts.

In the typing rules for processes (Fig. 4), presented as a set of right and left rules (the former identifying how to *offer* a session of a given type and the latter how to use such a session), we highlight the rules for dependently-typed communication and monadic elimination (for type-checking purposes we annotate constructs with the respective dependent type – this is akin to functional type theories). To offer a session $c{:}\exists x{:}\tau.A$ we send a term M of type τ and then offer a session $c{:}A\{M/x\}$; dually, to use such a session we perform an input along c, bound to x in Q, warranting a use of c as a session of (open) type A. The rules for the universal are dual. Offering a session $c{:}\forall x{:}\tau.A$ entails receiving on c a term of type τ and offering $c{:}A$. Using a session of such a type requires sending along c a term M of type τ, warranting the use of c as a session of type $A\{M/x\}$.

The rule for the monadic elimination form requires that the term M be of the appropriate monadic type and that the provided channels $\overline{u_j}$ and $\overline{y_i}$ adhere to the typing specified in M's type. Under these conditions, the process Q may then use the session c as session A. The type conversion rules reflect session type definitional equality in typing.

$$(\mathit{\Pi I})\quad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash M : \sigma}{\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma} \qquad (\mathit{\Pi E})\quad \frac{\Psi \vdash M : \Pi x{:}\tau.\sigma \quad \Psi \vdash N : \tau}{\Psi \vdash M\, N : \sigma\{N/x\}}$$

$$(\{\}I)\quad \frac{\forall i,j.\Psi \vdash A_i, B_j :: \mathsf{stype} \quad \Psi; \overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash P :: c{:}A}{\Psi \vdash \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} : \{\overline{u_j{:}B_j}; \overline{d_i : A_i} \vdash c{:}A\}} \qquad (\mathsf{Conv})\quad \frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \mathsf{type}}{\Psi \vdash M : \sigma}$$

Fig. 3. Typing for Terms (Excerpt – See Appendix A.4)

$$(\exists\mathsf{R})\quad \frac{\Psi \vdash M{:}\tau \quad \Psi; \Gamma; \Delta \vdash P :: c{:}A\{M/x\}}{\Psi; \Gamma; \Delta \vdash c\langle M\rangle_{\exists x:\tau.A}.P :: c{:}\exists x{:}\tau.A} \qquad (\exists\mathsf{L})\quad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau\ ;\ \Gamma; \Delta, c{:}A \vdash Q :: d{:}D}{\Psi\ ;\ \Gamma; \Delta, c{:}\exists x{:}\tau.A \vdash c(x{:}\tau).Q :: d{:}D}$$

$$(\forall\mathsf{R})\quad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau\ ;\ \Gamma; \Delta \vdash P :: c{:}A}{\Psi; \Gamma; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A} \qquad (\forall\mathsf{L})\quad \frac{\Psi \vdash M{:}\tau \quad \Psi; \Gamma; \Delta, c{:}A\{M/x\} \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta, c{:}\forall x{:}\tau.A \vdash c\langle M\rangle_{\forall x:\tau.A}.Q :: d{:}D}$$

$$(\{\}E)\quad \frac{\Delta' = \overline{d_i : B_i} \quad \overline{u_j{:}C_j} \subseteq \Gamma \quad \Psi \vdash M : \{\overline{u_j{:}C_j}; \overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Gamma; \Delta, c{:}A \vdash Q :: z{:}C}{\Psi; \Gamma; \Delta', \Delta \vdash c \leftarrow M \leftarrow \overline{u_j}; \overline{y_i}; Q :: z{:}C}$$

$$(\mathsf{ConvR})\quad \frac{\Psi; \Gamma; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \mathsf{stype}}{\Psi; \Gamma; \Delta \vdash P :: z{:}B} \qquad (\mathsf{ConvL})\quad \frac{\Psi; \Gamma'; \Delta' \vdash P :: z{:}A \quad \Psi; \Gamma'; \Delta' = \Psi; \Gamma; \Delta}{\Psi; \Gamma; \Delta \vdash P :: z{:}A}$$

$$(\mathsf{cut})\ \frac{\Psi; \Gamma; \Delta \vdash P :: c{:}A \quad \Psi; \Gamma; \Delta', c{:}A \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta, \Delta' \vdash (\boldsymbol{\nu} c)(P \mid Q) :: d{:}D}$$

Fig. 4. Typing for Processes (Excerpt – See Appendix A.5)

Definitional Equality. The crux of any dependent type theory lies in its *definitional equality.* Type equality relies on equality of terms which, by including the monadic construct, necessarily relies on a notion of *process* equality.

Our presentation of an intensional definitional equality of terms follows that of [12], where we consider an intrinsically typed relation, including β and η conversion (similarly for type equality which includes β and η principles for the type-level λ-abstractions). An excerpt of the rules for term equality is given in Fig. 5. The remaining rules are congruence rules and closure under symmetry, reflexivity and transitivity. Rule $(\mathsf{TMEq}\beta)$ captures the β-reduction, identifying a λ-abstraction applied to an argument with the substitution of the argument in the function body (typed with the appropriately substituted type). We highlight rule $(\mathsf{TMEq}\{\}\eta)$, which codifies a general η-like principle for arbitrary terms of monadic type: We form a monadic term that applies the monadic elimination form to M, forwarding the result along the appropriate channel, which becomes a term equivalent to M.

$$\textsf{(TMEq}\beta\textsf{)}\ \frac{\Psi \vdash \tau :: \textsf{type} \quad \Psi, x{:}\tau \vdash M : \sigma \quad \Psi \vdash N : \tau}{\Psi \vdash (\lambda x{:}\tau.M)\, N = M\{N/x\} : \sigma\{N/x\}} \qquad \textsf{(TMEq}\eta\textsf{)}\ \frac{\Psi \vdash M : \Pi x{:}\tau.\sigma \quad x \notin fv(M)}{\Psi \vdash \lambda x{:}\tau.M\, x = M : \Pi x{:}\tau.\sigma}$$

$$\textsf{(TMEq}\{\}\eta\textsf{)}\ \frac{\Psi \vdash M : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}}{\Psi \vdash \{c \leftarrow (y \leftarrow M; \overline{u_j}; \overline{d_i}; [y \leftrightarrow c]) \leftarrow \overline{u_j}; \overline{d_i}\} = M : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}}$$

Fig. 5. Definitional Equality of Terms (Excerpt – See Appendix A.9)

$$\textsf{(PEqRed)}\ \frac{\Psi; \Gamma; \Delta \vdash P :: z{:}A \quad P \rightarrow Q \quad \Psi; \Gamma; \Delta \vdash Q :: z{:}A}{\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A}$$

$$\textsf{(PEq}\forall\eta\textsf{)}\ \frac{}{\Psi; \Gamma; d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c] = [d \leftrightarrow c] :: c{:}\forall x{:}\tau.A}$$

$$\textsf{(PEqCC}\forall\textsf{)}\ \frac{\Psi; \Gamma; \Delta \vdash P :: d{:}B \quad \Psi, x{:}\tau; \Gamma; \Delta', d{:}B \vdash Q :: c{:}A}{\Psi; \Gamma; \Delta, \Delta' \vdash (\boldsymbol{\nu} d)(P \mid c(x).Q) = c(x).(\boldsymbol{\nu} d)(P \mid Q) :: c{:}\forall x{:}\tau.A}$$

Fig. 6. Definitional Equality of Processes (Excerpt – See Appendix A.10)

Definitional equality of processes is summarised in Fig. 6. We rely on process reduction defined below. Definitional equality of processes consists of the usual congruence rules, (typed) reductions and the commutting conversions of linear logic and η-like principles, which allows for forwarding actions to be equated with the primitive syntactic forwarding construct. Commutting conversions amount to sound observational equivalences between processes [24], given that session composition requires name restriction (embodied by the (cut) rule): In rule (PEqCC∀), either process can only be interacted with via channel c and so postponing actions of P to after the input on c (when reading the equality from left to right) cannot impact the process' observable behaviours. While P can in general interact with sessions in Δ (or with Q), these interactions are unobservable due to hiding in the (cut) rule.

Operational Semantics. The operational semantics for the λ-calculus is standard, noting that no reduction can take place inside monadic terms. The operational (reduction) semantics for processes is presented below where we omit closure under structural congruence and the standard congruence rules [4,27,31]. The last rule defines spawning a process in a monadic term.

$$\begin{array}{ll}
c\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{M/x\} & \overline{c}\langle x\rangle.P \mid c(x).Q \rightarrow (\boldsymbol{\nu} x)(P \mid Q) \\
!c(x).P \mid \overline{c}\langle x\rangle.Q \rightarrow\, !c(x).P \mid (\boldsymbol{\nu} x)(P \mid Q) & c.\textsf{case}\{\overline{l_i \Rightarrow P_i}\} \mid c.l_j; Q \rightarrow P_j \mid Q \quad (l_j \in \overline{l_i}) \\
(\boldsymbol{\nu} c)(P \mid [c \leftrightarrow d]) \rightarrow P\{d/c\} & c \leftarrow \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} \leftarrow \overline{u_j}; \overline{d_i}; Q \rightarrow (\boldsymbol{\nu} c)(P \mid Q)
\end{array}$$

2.3 Example – Reasoning about Processes using Dependent Types

The use of type indices (i.e. type families) in dependently typed frameworks adds information to types to produce more refined specifications. Our framework enables us to do this at the level of session types.

Consider a session type that "counts down" on a natural number (we assume inductive definitions and dependent pattern matching in the style of [23]):

$$\begin{array}{ll} \mathsf{countDown} & :: \mathit{\Pi x}{:}\mathsf{Nat}.\mathsf{stype} \\ \mathsf{countDown}\,(\mathsf{succ}(n)) & = \exists y{:}\mathsf{Nat}.\mathsf{countDown}(n) \\ \mathsf{countDown}\;\;\mathsf{z} & = \mathbf{1} \end{array}$$

The type family $\mathsf{countDown}(n)$ denotes a session type that emits exactly n numbers and then terminates. We can now write a (dependently-typed) function that produces processes with the appropriate type, given a starting value:

$$\begin{array}{ll} \mathsf{counter} & : \mathit{\Pi x}{:}\mathsf{Nat}.\{\mathsf{countDown}(x)\} \\ \mathsf{counter}\;\;(\mathsf{succ}(n)) & = \{c \leftarrow c\langle \mathsf{succ}(n)\rangle.\, d \leftarrow \mathsf{counter}(n); [d \leftrightarrow c]\} \\ \mathsf{counter}\;\;\mathsf{z} & = \{c \leftarrow \mathbf{0}\} \end{array}$$

Note how the type of counter, through the type family countDown, allows us to specify exactly the number of times a value is sent. This is in sharp contrast with existing recursive (or inductive/coinductive [19,32]) session types, where one may only specify the general iterative nature of the behaviour (e.g. "send a number and then recurse or terminate").

The example above relies on session type indexing in order to provide additional static guarantees about processes (and the functions that generate them). An alternative way is to consider "simply-typed" programs and then *prove* that they satisfy the desired properties, using the language itself. Consider a simply-typed version of the counter above described as an inductive session type:

$$\begin{array}{l} \mathsf{simpleCounterT} :: \mathsf{stype} \\ \mathsf{simpleCounterT} = \oplus\{\mathsf{dec} : \mathsf{Nat} \wedge \mathsf{simpleCounterT}, \mathsf{done} : \mathbf{1}\} \end{array}$$

There are many processes that correctly implement such a type, given that the type merely dictates that the session outputs a natural number and recurses (modulo the dec and done messages to signal which branch of the internal choice is taken). A function that produces processes implementing such a session, mirroring those generated by the counter function above, is:

$$\begin{array}{ll} \mathsf{simpleCounter} & : \mathsf{Nat} \rightarrow \{\mathsf{simpleCounterT}\} \\ \mathsf{simpleCounter}\;\;(\mathsf{succ}(n)) & = \{c \leftarrow c.\mathsf{dec}; (\boldsymbol{\nu} d)(d\langle \mathsf{succ}(n)\rangle.\mathbf{0} \mid d(x).c\langle x\rangle. \\ & \quad d \leftarrow \mathsf{simpleCounter}(n); [d \leftrightarrow c]\} \\ \mathsf{simpleCounter}\;\;\;\mathsf{z} & = \{c \leftarrow c.\mathsf{done}; \mathbf{0}\} \end{array}$$

The process generated by simpleCounter, after emiting the dec label, spawns a process in parallel that sends the appropriate number, which is received by the parallel thread and then sent along the session c. Despite its simplicity, this example embodies a general pattern where a computation is spawned in parallel (itself potentially spawning many other threads) and the main thread then waits for the result before proceeding.

While such a process is typable in most session typing frameworks, our theory enables us to *prove* that the counter implementation above indeed counts down from a given number by defining an appropriate (inductive) type family, indexed by *monadic* values (i.e. processes):

$$\begin{array}{lll}
\mathsf{corrCount} & :: & \Pi x{:}\mathsf{Nat}.\Pi y{:}\{\mathsf{simpleCounterT}\}.\mathsf{type} \\
\mathsf{corr}_z & : & \mathsf{corrCount}\,\mathsf{z}\,\{c \leftarrow c.\mathsf{done}; \mathbf{0}\} \\
\mathsf{corr}_n & : & \Pi n{:}\mathsf{Nat}.\Pi P{:}\{\mathsf{simpleCounterT}\}.\mathsf{corrCount}\,n\,P \rightarrow \\
 & & \mathsf{corrCount}\,(\mathsf{succ}(n))\,\{c \leftarrow c.\mathsf{dec}; c\langle \mathsf{succ}(n)\rangle.d \leftarrow P; [d \leftrightarrow c]\}
\end{array}$$

The type family corrCount, indexed by a natural number and a monadic value implementing the session type simpleCounter, is defined via two constructors: corr_z, which specifies that a correct 0 counter emits the done label and terminates; and corr_n, which given a monadic value P that is a correct n-counter, defines that a correct $(n+1)$-counter emits $n+1$ and then proceeds as P (modulo the label emission bookkeeping).

The proof of correctness of the simpleCounter function above is no more than a function of type $\Pi n{:}\mathsf{Nat}.\mathsf{corrCount}\,n\,(\mathsf{simpleCounter}(n))$, defined below:

$$\begin{array}{lll}
\mathsf{prf} & & : \Pi n{:}\mathsf{Nat}.\mathsf{corrCount}\,n\,(\mathsf{simpleCounter}(n)) \\
\mathsf{prf} & \mathsf{z} & = \mathsf{corr}_z \\
\mathsf{prf} & (\mathsf{succ}(n)) & = \mathsf{corr}_n\,n\,(\mathsf{simpleCounter}(n))\,(\mathsf{prf}\,n)
\end{array}$$

Note that in this scenario, the processes that index the corrCount type family are not syntactically equal to those generated by simpleCounter, but rather *definitionally* equal.

Typically, the processes that index such correctness specifications tend to be distilled versions of the actual implementations, which often perform some additional internal computation or communication steps. Since our notion of definitional equality of processes includes reduction (and also commuting conversions which account for type-preserving shuffling of internal communication actions [28]), the type conversion mechanism allows us to use the techniques described above to generally reason about specification conformance.

We may also consider a variant of the example above which does not force outputs to match precisely with the type index:

$$\begin{array}{ll}
\mathsf{countDown}' & :: \Pi x{:}\mathsf{Nat}.\mathsf{stype} \\
\mathsf{countDown}'\,(\mathsf{succ}(n)) & = \exists y{:}\mathsf{Nat}.\mathsf{countDown}'(n) \\
\mathsf{countDown}'\;\mathsf{z} & = \mathbf{1}
\end{array}$$

The type $\mathsf{countDown}'\,n$ will still require n outputs to be performed, but unlike with countDown we do not enforce a relation between the iteration and the number being sent. An implementation of such a type is given below, using fundamentally the same code as for counter:

$$\begin{array}{lll}
\mathsf{counter}' & & : \Pi x{:}\mathsf{Nat}.\{\mathsf{countDown}'(x)\} \\
\mathsf{counter}' & (\mathsf{succ}(n)) & = \{c \leftarrow c\langle \mathsf{succ}(n)\rangle. \\
 & & \quad\; d \leftarrow \mathsf{counter}'(n); \\
 & & \quad\; [d \leftrightarrow c]\} \\
\mathsf{counter}' & \mathsf{z} & = \{c \leftarrow \mathbf{0}\}
\end{array}$$

We may then use an heterogeneous equality (a special case of the so-called *John Major equality* [20]) of the form

$$\begin{array}{lll}\mathsf{JMEq} & :: & \Pi A{:}\mathsf{stype}.\Pi B{:}\mathsf{stype}.\Pi x{:}\{A\}.\Pi y{:}\{B\}.\mathsf{type} \\ \mathsf{JMEqRefl} & : & \lambda A{:}\mathsf{stype}.\lambda x{:}\{A\}.\mathsf{JMEq}\,A\,A\,x\,x\end{array}$$

to inductively show that the processes produced by $\mathsf{counter}$ and $\mathsf{counter}'$ are indeed the same.

$$\begin{array}{l}\mathsf{eqs} : \Pi n{:}\mathsf{Nat}.\mathsf{JMEq}\,(\mathsf{countDown}(n))\,(\mathsf{countDown}'(n))\,(\mathsf{counter}(n))\,(\mathsf{counter}'(n)) \\ \mathsf{eqs}\,z = \mathsf{JMEqRefl}\,\mathbf{1}\,\{c \leftarrow \mathbf{0}\} \\ \mathsf{eqs}\,(\mathsf{succ}(n)) = \mathsf{case}\,(\mathsf{eqs}\,n)\,\mathsf{of}\,\{\_ \Rightarrow \mathsf{JMEqRefl}\,(\mathsf{countDown}(\mathsf{succ}(n))) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\mathsf{counter}(\mathsf{succ}(n))))\}\end{array}$$

We note that the example above makes extensive use of dependent pattern matching, using some implicit assumptions on its behaviour that have not been formalised in this paper and are left for future work.

2.4 Type Soundness of the Framework

The main goal of this section is to present type soundness of our framework through a subject reduction result. We also show that our theory guarantees progress for terms and processes. The development requires a series of auxiliary results (detailed in Appendix B) pertaining to the functional and process layers which are ultimately needed to produce the inversion properties necessary to establish subject reduction. We note that strong normalisation results for linear-logic based session processes are known in the literature [3,32,28], even in the presence of impredicative polymorphism, restricted corecursion and higher-order data. Such results are directly applicable to our work using appropriate semantics preserving type erasures.

In the remainder we often write $\Psi \vdash \mathcal{J}$ to stand for a well-formedness, typing or definitional equality judgment of the appropriate form. Similarly for $\Psi; \Gamma; \Delta \vdash \mathcal{J}$. We begin with the substitution property, which naturally holds for both layers, noting that the dependently typed nature of the framework requires substitution in both contexts, terms and in types.

Lemma 2.1 (Substitution). *Let* $\Psi \vdash M : \tau$*:*

1. *If* $\Psi, x{:}\tau, \Psi' \vdash \mathcal{J}$ *then* $\Psi, \Psi'\{M/x\} \vdash \mathcal{J}\{M/x\}$*;*
2. *If* $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash \mathcal{J}$ *then* $\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash \mathcal{J}\{M/x\}$

Combining substitution with a form of functionality for typing (i.e. that substitution of equal terms in a well-typed term produces equal terms) and for equality (i.e. that substitution of equal terms in a definitional equality proof produces equal terms), we can establish validity for typing and equality, which is a form of internal soundness of the type theory stating that judgments are consistent across the different levels of the theory.

Lemma 2.2 (Validity for Typing). (1) *If* $\Psi \vdash \tau :: K$ *or* $\Psi \vdash A :: K$ *then* $\Psi \vdash K$*;* (2) *If* $\Psi \vdash M : \tau$ *then* $\Psi \vdash \tau :: \mathsf{type}$*; and* (3) *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *then* $\Psi \vdash A :: \mathsf{stype}$.

Lemma 2.3 (Validity for Equality).

1. *If* $\Psi \vdash M = N : \tau$ *then* $\Psi \vdash M : \tau$*,* $\Psi \vdash N : \tau$ *and* $\Psi \vdash \tau :: \mathsf{type}$
2. *If* $\Psi \vdash \tau = \sigma :: K$ *then* $\Psi \vdash \tau :: K$*,* $\Psi \vdash \sigma :: K$ *and* $\Psi \vdash K$
3. *If* $\Psi \vdash A = B :: K$ *then* $\Psi \vdash A :: K$*,* $\Psi \vdash B :: K$ *and* $\Psi \vdash K$
4. *If* $\Psi \vdash K = K'$ *then* $\Psi \vdash K$ *and* $\Psi \vdash K'$
5. *If* $\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$ *then* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$*,* $\Psi; \Gamma; \Delta \vdash Q :: z{:}A$ *and* $\Psi \vdash A :: \mathsf{stype}$

With these results we establish the appropriate inversion and injectivity properties which then enable us to show unicity of types (and kinds).

Theorem 2.4 (Unicity of Types and Kinds).

1. *If* $\Psi \vdash M : \tau$ *and* $\Psi \vdash M : \tau'$ *then* $\Psi \vdash \tau = \tau' :: \mathsf{type}$
2. *If* $\Psi \vdash \tau :: K$ *and* $\Psi \vdash \tau :: K'$ *then* $\Psi \vdash K = K'$
3. *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *and* $\Psi; \Gamma; \Delta \vdash P :: z{:}A'$ *then* $\Psi \vdash A = A' :: \mathsf{stype}$
4. *If* $\Psi \vdash A :: K$ *and* $\Psi \vdash A :: K'$ *then* $\Psi \vdash K = K'$

All the results above, combined with the process-level properties established in [29,28,5] enable us to show the following:

Theorem 2.5 (Subject Reduction – Terms). *If* $\Psi \vdash M : \tau$ *and* $M \to M'$ *then* $\Psi \vdash M' : \tau$

Theorem 2.6 (Subject Reduction – Processes). *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *and* $P \to P'$ *then* $\exists Q$ *such that* $P' \equiv Q$ *and* $\Psi; \Gamma; \Delta \vdash Q :: z{:}A$

Theorem 2.7 (Progress – Terms). *If* $\Psi \vdash M : \tau$ *then either* M *is a value or* $M \to M'$

As common in logical-based session type theories, typing enforces a strong notion of *global* progress which states that closed processes that are waiting to perform communication actions cannot get stuck (this relies on a notion of *live* process, defined as $\mathsf{live}(P)$ iff $P \equiv (\boldsymbol{\nu}\tilde{n})(\pi.Q \mid R)$ for some process R, sequence of names $\tilde{n}$ and a non-replicated guarded process $\pi.Q$). We note that the restricted typing for P is without loss of generality, due to the (cut) rule.

Theorem 2.8 (Progress – Processes). *If* $\Psi; \cdot; \cdot \vdash P :: c{:}\mathbf{1}$ *and* $\mathsf{live}(P)$ *then* $\exists Q$ *such that* $P \to Q$

3 Embedding the Functional Layer in the Process Layer

Having introduced our type theory and showcased some of its informal expressiveness in terms of the ability to specify and *statically* verify true data dependent protocols, as well as the ability to prove properties of processes, we now develop a formal expressiveness result for our theory, showing that the process level type constructs are able to encode the dependently-typed functional layer, faithfully preserving type dependencies.

Specifically, we show that (1) the type-level constructs in the functional layer can be represented by those in the process layer combined with the contextual monad type, and (2) all term level constructs can be represented by session-typed processes that exchange monadic values. Thus, we show that both λ-abstraction and application can be eliminated while still preserving non-trivial type dependencies. Crucially, we note that the monadic construct *cannot* be fully eliminated due to the cross-layer nature of session type dependencies: In the process layer, simply-kinded dependent types (i.e. types with kind stype) are of the form $\forall x{:}\tau.A$ where τ is of kind type and A of kind stype (where x may occur). Operationally, such a session denotes an input of some term M of type τ with a continuation of type $A\{M/x\}$. Thus, to faithfully encode type dependencies we cannot represent such a type with a non-dependently typed input (e.g. a type of the form $A \multimap B$).

3.1 The Embedding

A first attempt. Given the observation above, a seemingly reasonable option would be to attempt an encoding that maintains monadic objects solely at the level of type indices and then exploits Girard's encoding [9] of function types $\tau \to \sigma$ as $!\llbracket\tau\rrbracket \to \llbracket\sigma\rrbracket$, which is adequate for session-typed processes [30]. Thus a candidate encoding for the type $\Pi x{:}\tau.\sigma$ would be $\forall x{:}\{\llbracket\tau\rrbracket\}.!\llbracket\tau\rrbracket \multimap \llbracket\sigma\rrbracket$, where $\llbracket - \rrbracket$ denotes our encoding on types. If we then consider the encoding at the level of terms, typing dictates the following (we write $\llbracket M\rrbracket_z$ for the process encoding of $M : \tau$, where z is the session channel along which one may observe the "result" of the encoding, typed with $\llbracket\tau\rrbracket$):

$$\begin{array}{ll}\llbracket\lambda x{:}\tau.M\rrbracket_z & \triangleq z(x).z(x').\llbracket M\rrbracket_z \\ \llbracket M\,N\rrbracket_z & \triangleq (\boldsymbol{\nu}x)(\llbracket M\rrbracket_x \mid x\langle\{\llbracket N\rrbracket_y\}\rangle.\overline{x}\langle x'\rangle.(!x'(y).\llbracket N\rrbracket_y \mid [x \leftrightarrow z])\end{array}$$

However, this candidate encoding breaks down once we consider definitional equality. Specifically, compositionality (i.e. the relationship between $\llbracket M\{N/x\}\rrbracket_z$ and the encoding of N substituted in that of M) requires us to relate $\llbracket M\{N/x\}\rrbracket_z$ with $(\boldsymbol{\nu}x)(\llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_y\}/x\} \mid !x'(y).\llbracket N\rrbracket_y)$, which relies on reasoning up-to *observational equivalence* of processes, a much stronger relation than our notion of definitional equality. Therefore it is *fundamentally* impossible for such an encoding to preserve our definitional equality, and thus it cannot preserve typing in the general case.

Kind:

$$\begin{array}{llll}
\llbracket \mathsf{type} \rrbracket & \triangleq \mathsf{stype} & \llbracket \mathsf{stype} \rrbracket & \triangleq \mathsf{stype} \\
\llbracket \Pi x{:}\tau.K \rrbracket & \triangleq \Pi x{:}\{\llbracket \tau \rrbracket\}.\llbracket K \rrbracket & \llbracket \Pi t :: K_1.K_2 \rrbracket & \triangleq \Pi t{::}\llbracket K_1 \rrbracket.\llbracket K_2 \rrbracket
\end{array}$$

Functional:

$$\begin{array}{llll}
\llbracket \Pi x{:}\tau.\sigma \rrbracket & \triangleq \forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket & \llbracket \{\overline{u_j{:}B_j}; \overline{d_i{:}B_i} \vdash c{:}A\} \rrbracket & \triangleq \overline{!\llbracket B_j \rrbracket} \multimap \overline{\llbracket B_i \rrbracket} \multimap \llbracket A \rrbracket \\
\llbracket \lambda x{:}\tau.\sigma \rrbracket & \triangleq \lambda x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket & \llbracket \tau\, M \rrbracket & \triangleq \llbracket \tau \rrbracket\, \{\llbracket M \rrbracket_c\} \\
\llbracket \lambda t{::}K.\tau \rrbracket & \triangleq \lambda t{::}\llbracket K \rrbracket.\llbracket \tau \rrbracket & \llbracket \tau\, \sigma \rrbracket & \triangleq \llbracket \tau \rrbracket\, \llbracket \sigma \rrbracket
\end{array}$$

Session:

$$\begin{array}{llll}
\llbracket \forall x{:}\tau.A \rrbracket & \triangleq \forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket A \rrbracket & \llbracket \exists x{:}\tau.A \rrbracket & \triangleq \exists x{:}\{\llbracket \tau \rrbracket\}.\llbracket A \rrbracket \\
\llbracket \lambda x{:}\tau.A \rrbracket & \triangleq \lambda x{:}\{\llbracket \tau \rrbracket\}.\llbracket A \rrbracket & \llbracket A\, M \rrbracket & \triangleq \llbracket A \rrbracket\, \{\llbracket M \rrbracket_c\}
\end{array}$$

Terms:

$$\begin{array}{ll}
\llbracket \lambda x{:}\tau.M \rrbracket_z \triangleq z(x{:}\{\llbracket \tau \rrbracket\}).\llbracket M \rrbracket_z & \llbracket M\, N \rrbracket_z \triangleq (\boldsymbol{\nu} x)(\llbracket M \rrbracket_x \mid x\langle \{\llbracket N \rrbracket_y\} \rangle.[x \leftrightarrow z]) \\
\llbracket x \rrbracket_z \triangleq y \leftarrow x; [y \leftrightarrow z] & \llbracket \{z \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} \rrbracket_z \triangleq z(u_0).\ldots.z(u_j).z(d_0).\ldots.z(d_n).\llbracket P \rrbracket
\end{array}$$

Processes:

$$\begin{array}{ll}
\llbracket (\boldsymbol{\nu} x)(P \mid Q) \rrbracket & \triangleq (\boldsymbol{\nu} x)(\llbracket P \rrbracket \mid \llbracket Q \rrbracket) \quad \llbracket \mathbf{0} \rrbracket \triangleq \mathbf{0} \quad \llbracket \overline{x}\langle y \rangle.(P \mid Q) \rrbracket \triangleq \overline{x}\langle y \rangle.(\llbracket P \rrbracket \mid \llbracket Q \rrbracket) \\
\llbracket x\langle M \rangle.P \rrbracket & \triangleq x\langle \{\llbracket M \rrbracket_y\} \rangle.\llbracket P \rrbracket \quad \llbracket x(y).P \rrbracket \triangleq x(y).\llbracket P \rrbracket \\
\llbracket c \leftarrow M \leftarrow \overline{u_j}; \overline{y_i}; Q \rrbracket & \triangleq (\boldsymbol{\nu} c)(\llbracket M \rrbracket_c \mid \overline{c}\langle v_1 \rangle.(\overline{u_1}\langle a_1 \rangle.[a_1 \leftrightarrow v_1] \mid \cdots \mid \\
& \quad \overline{c}\langle d_1 \rangle.([y_1 \leftrightarrow d_1] \mid \cdots \mid \overline{c}\langle d_n \rangle.([y_n \leftrightarrow d_n] \mid \llbracket Q \rrbracket) \ldots)
\end{array}$$

Fig. 7. An embedding of dependent functions into processes

A faithful embedding. We now develop our embedding of the functional layer into the process layer which is compatible with definitional equality. Our target calculus is reminiscent of a higher-order (in the sense of higher-order processes [25]) session calculus [21]. Our encoding $\llbracket - \rrbracket$ is inductively defined on kinds, types, session types, terms and processes. As usual in process encodings of the λ-calculus, the encoding of a term M is indexed by a result channel z, written $\llbracket M \rrbracket_z$, where the behaviour of M may be observed.

The embedding is presented in Fig. 7, noting that the encoding extends straightforwardly to typing contexts, where functional contexts $\Psi, x{:}\tau$ are mapped to $\{\llbracket \Psi \rrbracket\}, x{:}\{\llbracket \tau \rrbracket\}$. The mapping of base kinds is straightforward. Dependent kinds $\Pi x{:}\tau.K$ rely on the monad for well-formedness and are encoded as (session) kinds of the form $\Pi x{:}\{\llbracket \tau \rrbracket\}.\llbracket K \rrbracket$. The higher-kinded types in the functional layer are translated to the corresponding type-level constructs of the process layer where all objects that must be type-kinded rely on the monad to satisfy this constraint. For instance, $\lambda x{:}\tau.\sigma$ is mapped to the session-type abstraction $\lambda x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket$ and the type-level application $\tau\, M$ is translated to $\llbracket \tau \rrbracket\, \{\llbracket M \rrbracket_c\}$. Given the observation above on embedding the dependent function type $\Pi x{:}\tau.\sigma$, we translate it directly to $\forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket$, that is, functions from τ to σ are mapped to sessions that input *processes* implementing $\llbracket \tau \rrbracket$ and then behave as $\llbracket \sigma \rrbracket$ accordingly. The encoding for monadic types simply realises the contextual nature of the monad by performing a sequence of inputs of the appropriate types (with the shared sessions being of ! type).

The mutually dependent nature of the framework requires us to extend the mapping to the process layer. Session types are mapped homomorphically (e.g. $[\![A \multimap B]\!] \triangleq [\![A]\!] \multimap [\![B]\!]$) with the exception of dependent inputs and outputs which rely on the monad, similarly for type-level functions and application.

The encoding of λ-terms is guided by the embedding for types: the abstraction $\lambda x{:}\tau.M$ is mapped to an input of a term of type $\{[\![\tau]\!]\}$ with continuation $[\![M]\!]_z$; application $M\,N$ is mapped to the composition of the encoding of M on a fresh name x with the corresponding output of $\{[\![N]\!]_y\}$, which is then forwarded to the result channel z; monadic expressions are translated to the appropriate sequence of inputs, as dictated by the translation of the monadic type; and, the translation of variables makes use of the monadic elimination form (since the encoding enforces variables to always be of monadic type) combined with forwarding to the appropriate result channel.

The mapping for processes is mostly homomorphic, using the monad constructor as needed. The only significant exception is the encoding for monadic elimination which must provide the encoded monadic term $[\![M]\!]_c$ with the necessary channels. Since the session calculus does not support communication of free names this is achieved by a sequence of outputs of fresh names combined with forwarding of the appropriate channel. To account for replicated sessions we must first trigger the replication via an output which is then forwarded accordingly.

We can illustrate our encoding via a simple example of an encoded function (we omit type annotations for conciseness):

$$\begin{array}{l}
[\![(\lambda x.x)\,(\lambda x.\lambda y.y)]\!]_z = (\boldsymbol{\nu} c)([\![\lambda x.x]\!]_c \mid c\langle\{[\![\lambda x.\lambda y.y]\!]_w\}\rangle.[c \leftrightarrow z]) = \\
\qquad (\boldsymbol{\nu} c)(c(x).y \leftarrow x; [y \leftrightarrow c] \mid c\langle\{w(x).w(y).d \leftarrow y; [d \leftrightarrow w]\}\rangle.[c \leftrightarrow z]) \\
\to^{+} z(x).z(y).d \leftarrow y; [d \leftrightarrow z] \;=\; [\![\lambda x.\lambda y.y]\!]_z
\end{array}$$

3.2 Properties of the Embedding

We now state the key properties satisfied by our embedding, ultimately resulting in type preservation and operational correspondence. For conciseness, in the statements below we list only the cases for terms and processes, omitting those for types and kinds (see Appendix C). The key property that is needed is a notion of compositionality, which unlike in the sketch above no longer falls outside of definitional equality.

Lemma 3.1 (Compositionality).

1. *$\Psi; \Gamma; \Delta \vdash [\![M\{N/x\}]\!]_z = [\![M]\!]_z\{\{[\![N]\!]_y\}/x\} :: z{:}[\![A\{N/x\}]\!]$*
2. *$\Psi; \Gamma; \Delta \vdash [\![P\{M/x\}]\!] :: z{:}[\![A\{M/x\}]\!]$ iff $\Psi; \Gamma; \Delta \vdash [\![P]\!]\{\{[\![M]\!]_c\}/x\} :: z{:}[\![A]\!]\{\{[\![M]\!]_c\}/x\}$*

Given the dependently typed nature of the framework, establishing the key properties of the encoding must be done simultaneously (relying on some auxiliary results – see Appendix C).

Theorem 3.2 (Preservation of Equality).

1. *If $\Psi \vdash M = N : \tau$ then $\{[\![\Psi]\!]\}; \cdot; \cdot \vdash [\![M]\!]_z = [\![N]\!]_z :: z{:}[\![\tau]\!]$*
2. *If $\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$ then $\{[\![\Psi]\!]\}; [\![\Gamma]\!]; [\![\Delta]\!] \vdash [\![P]\!] = [\![Q]\!] :: z{:}[\![A]\!]$*

Theorem 3.3 (Preservation of Typing).

1. *If* $\Psi \vdash M : \tau$ *then* $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_z :: z{:}\llbracket\tau\rrbracket$
2. *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *then* $\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P \rrbracket :: z{:}\llbracket A\rrbracket$

Theorem 3.4 (Operational Correspondence). *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *and* $\Psi \vdash M : \tau$ *then:*

1. *(a) If* $P \to P'$ *then* $\llbracket P \rrbracket \to^+ Q$ *with* $\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash Q = \llbracket P' \rrbracket :: z{:}\llbracket A\rrbracket$ *and*
   (b) if $\llbracket P \rrbracket \to P'$ *then* $P \to^+ Q$ *with* $\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash P' = \llbracket Q \rrbracket :: z{:}\llbracket A\rrbracket$
2. *(a) If* $M \to M'$ *then* $\llbracket M \rrbracket_z \to^+ N$ *with* $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash N = \llbracket M' \rrbracket_z :: z{:}\llbracket\tau\rrbracket$ *and*
   (b) if $\llbracket M \rrbracket_z \to P$ *then* $M \to N$ *with* $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket N \rrbracket_z = P :: z{:}\llbracket\tau\rrbracket$

In Theorem 3.4, (a) is commonly referred to as operational completeness, with (b) establishing soundness. As exemplified above, our encoding satisfies a very precise operational correspondence with the original λ-terms.

4 Related and Future Work

Enriching Session Types via Type Structure. Exploiting the linear logical foundations of session types, [27] considers a form of value dependencies where session types can state properties of exchanged data values, while the work [31] introduces the contextual monad in a simply-typed setting. Our development not only subsumes these two works, but goes beyond simple value dependencies by extending to a richer type structure and integrating dependencies with the contextual monad. Recently, [1] considers a non-conservative extension of linear logic-based session types with sharing, allowing true non-determinism. Their work includes dependent quantifications with shared channels, but their type syntax does *not* include free type variables, so the actual type dependencies do not arise (see [1, 37:8]). Thus none of the examples in this paper can be represented in [1]. The work [17] studies gradual session types. To the best of our knowledge, the main example in [17, § 2] is *statically* representable in our framework as in the example of § 1, where protocol actions depend on values that are communicated (or passed as function arguments).

In the context of multiparty session types, the theory of multiparty indexed session types is studied in [7], and implemented in a protocol description language [22]. The main aim of these works is to use indexed types to represent an arbitrary number of session *participants*. The work [33] extends [27] to multiparty sessions in order to treat value dependency across multiple participants. Extending our framework to multiparty [16] or non-logic based session types [15] is an interesting future topic.

Combining Linear and Dependent Types. Many works have studied the various challenges of integrating linearity in dependent functional type theories. We focus on the most closely related works. The work [6] introduced the Linear Logical Framework (LLF), integrating linearity with the LF [11] type theory, which was later extended to the Concurrent Logical Framework (CLF) [35], accounting for further linear connectives. Their theory is representable in our framework through the contextual monad (encompassing full intuitionistic linear logic), depending on linearly-typed processes that can express dependently typed functions (§ 3).

The work of [18] integrates linearity with type dependencies by extending LNL [2]. Their work is aimed at reasoning about imperative programs using a form of Hoare triples, requiring features that we do not study in this work such has proof irrelevance and computationally irrelevant quantification. Formally, their type theory is extensional which introduces significant technical differences from our intensional type theory, such as a realisability model in the style of NuPRL [10] to establish consistency.

Recently, [8] proposed an extension of LLF with first-class contexts (which may contain both linear and unrestricted hypotheses). While the contextual aspects of their theory are reminiscent of our contextual monad, their framework differs significantly from ours, since it is designed to enable higher-order abstract syntax (commonplace in the LF family of type theories), focusing on a type system for canonical LF objects with a meta-language that includes contexts and context manipulation. They do not consider additives since their integration with first-class contexts can break canonicity.

While none of the above works considers processes as primitive, their techniques should be useful for, e.g. developing algorithmic type-checking and integrating inductive and coinductive session types based on [28,32,19].

Dependent Types and Higher-Order π-calculus. The work [37] studies a form of dependent types where the type of processes takes the form of a mapping Δ from channels x to channel types T representing an interface of process P. The dependency is specified as $\Pi(x{:}T)\Delta$, representing a channel abstraction of the environment. This notion is extended to an existential channel dependency type $\Sigma(x{:}T)\Delta$ to address fresh name creation [36,13]. Combining our process monad with dependent types can be regarded as an "interface" which describes explicit channel usages for processes. The main differences are (1) our dependent types are more general, treating full dependent families including terms and processes in types, while [37,36,13] study only channel dependency to environments (i.e. neither terms nor processes appear in types, only channels); and (2) our calculus emits only fresh names, not needing to handle the complex scoping mechanism treated in [36,13]. In this sense, the process monad provides an elegant framework to handle higher-order computations and assign non-trivial types to processes.

Acknowledgements. The authors would like to thank the anonymous reviews for their comments and suggestions. This work is partially supported by EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1.

References

1. Balzer, S., Pfenning, F.: Manifest sharing with session types. PACMPL 1(ICFP), 37:1–37:29 (2017)

2. Benton, N.: A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In: CSL. pp. 121–135 (1994)
3. Caires, L., Pérez, J.A., Pfenning, F., Toninho, B.: Behavioral polymorphism and parametricity in session-based communication. In: ESOP 2013. pp. 330–349 (2013)
4. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: CONCUR 2010. pp. 222–236 (2010)
5. Caires, L., Pfenning, F., Toninho, B.: Linear logic propositions as session types. Mathematical Structures in Computer Science 26(3), 367–423 (2016)
6. Cervesato, I., Pfenning, F.: A linear logical framework. Inf. Comput. 179(1), 19–75 (2002)
7. Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Logical Methods in Computer Science 8(4) (2012), `http://dx.doi.org/10.2168/LMCS-8(4:6)2012`
8. Georges, A.L., Murawska, A., Otis, S., Pientka, B.: LINCX: A linear logical framework with first-class contexts. In: ESOP. pp. 530–555 (2017)
9. Girard, J.: Linear logic. Theor. Comput. Sci. 50, 1–102 (1987)
10. Harper, R.: Constructing type systems over an operational semantics. Journal of Symbolic Computation 14(1), 71 – 84 (1992)
11. Harper, R., Honsell, F., Plotkin, G.D.: A framework for defining logics. J. ACM 40(1), 143–184 (1993)
12. Harper, R., Pfenning, F.: On equivalence and canonical forms in the LF type theory. ACM Trans. Comput. Log. 6(1), 61–101 (2005)
13. Hennessy, M., Rathke, J., Yoshida, N.: safeDpi: a language for controlling mobile code. Acta Inf. 42(4-5), 227–290 (2005)
14. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP'98. pp. 122–138 (1998)
15. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: ESOP'98. vol. 1381, pp. 22–138 (1998)
16. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL'08. pp. 273–284 (2008)
17. Igarashi, A., Thiemann, P., Vasconcelos, V.T., Wadler, P.: Gradual session types. PACMPL 1(ICFP), 38:1–38:28 (2017)
18. Krishnaswami, N.R., Pradic, P., Benton, N.: Integrating linear and dependent types. In: POPL'15. pp. 17–30 (2015)
19. Lindley, S., Morris, J.G.: Talking bananas: structural recursion for session types. In: ICFP 2016. pp. 434–447 (2016)
20. McBride, C.: Elimination with a motive. In: TYPES 2000,. pp. 197–216 (2000)
21. Mostrous, D., Yoshida, N.: Two session typing systems for higher-order mobile processes. In: TLCA07. pp. 321–335 (2007)
22. Ng, N., Yoshida, N.: Pabble: parameterised Scribble. Service Oriented Computing and Applications 9(3-4), 269–284 (2015)
23. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology (2007)
24. Pérez, J.A., Caires, L., Pfenning, F., Toninho, B.: Linear logical relations for session-based concurrency. In: ESOP. pp. 539–558 (2012)
25. Sangiorgi, D., Walker, D.: The pi-calculus: A theory of mobile processes. C.U.P (2001)

26. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: PARLE'94. pp. 398–413 (1994)
27. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: PPDP'11. pp. 161–172 (2011)
28. Toninho, B.: A Logical Foundation for Session-based Concurrent Computation. Ph.D. thesis, Carnegie Mellon University and New University of Lisbon (2015)
29. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. Tech. Rep. CMU-CS-11-139, School of Computer Science, Carnegie Mellon University (2011)
30. Toninho, B., Caires, L., Pfenning, F.: Functions as session-typed processes. In: FOSSACS 2012. pp. 346–360 (2012)
31. Toninho, B., Caires, L., Pfenning, F.: Higher-order processes, functions, and sessions: A monadic integration. In: ESOP. pp. 350–369 (2013)
32. Toninho, B., Caires, L., Pfenning, F.: Corecursion and non-divergence in session-typed processes. In: TGC 2014. pp. 159–175 (2014)
33. Toninho, B., Yoshida, N.: Certifying data in multiparty session types. Journal of Logical and Algebraic Methods in Programming 90(C), 61–83 (2017)
34. Toninho, B., Yoshida, N.: Depending on session-typed processes. In: FoSSaCS (2018), to Appear
35. Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: A concurrent logical framework: The propositional fragment. In: TYPES'03. pp. 355–377 (2003)
36. Yoshida, N.: Channel dependent types for higher-order mobile processes. In: POPL'04. pp. 147–160 (2004)
37. Yoshida, N., Hennessy, M.: Assigning types to processes. Inf. Comput. 174(2), 143–179 (2002)

A Appendix – Dependently-typed Calculus

A.1 Complete Rules for Dependently-Typed System

We recall the meaning of the several judgments of our type theory:

$\Psi \vdash$	Context Ψ is well-formed.
$\Psi \vdash K$	K is a kind in context Ψ.
$\Psi \vdash \tau :: K$	τ is a (functional) type of kind K in context Ψ.
$\Psi \vdash A :: K$	A is a session type of kind K in context Ψ.
$\Psi \vdash M : \tau$	M has type τ in context Ψ.
$\Psi; \Gamma; \Delta \vdash P :: z{:}A$	P offers session $z{:}A$ when composed with processes according to Γ and Δ in context Ψ.
$\Psi \vdash K_1 = K_2$	Kinds K_1 and K_2 are equal.
$\Psi \vdash \tau = \sigma :: K$	Types τ and σ are equal of kind K.
$\Psi \vdash A = B :: K$	Session types A and B are equal of kind K.
$\Psi \vdash M = N : \tau$	Terms M and N are equal of type τ.
$\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$	Processes P and Q are equal with typing $z{:}A$.

Well-formed Contexts We write $\cdot$ for the empty context. We write $\Psi, x{:}\tau$ for the extension of context Ψ with the binding $x{:}\tau$. We assume that x does not occur in Ψ. We use a similar notation for the session typing contexts Δ and Γ.

$$\frac{}{\cdot \vdash} \qquad \frac{\Psi \vdash \quad \Psi \vdash \tau :: \mathsf{type}}{\Psi, x{:}\tau \vdash} \qquad \frac{\Psi \vdash \quad \Psi \vdash K}{\Psi, t{::}K \vdash} \qquad \frac{\Psi \vdash \quad \Psi; \Delta \vdash \quad \Psi \vdash A :: \mathsf{stype}}{\Psi; \Delta, x{:}A \vdash}$$

$$\frac{\Psi \vdash \quad \Psi; \Gamma \vdash \quad \Psi \vdash A :: \mathsf{stype}}{\Psi; \Gamma, x{:}A \vdash}$$

Well-formed Kinds

$$\frac{\Psi \vdash}{\Psi \vdash \mathsf{type}} \qquad \frac{\Psi \vdash}{\Psi \vdash \mathsf{stype}} \qquad \frac{\Psi, x{:}\tau \vdash K \quad \Psi \vdash \tau :: \mathsf{type}}{\Psi \vdash \Pi x{:}\tau.K} \qquad \frac{\Psi, x{:}\tau \vdash K \quad \Psi \vdash \tau :: \mathsf{stype}}{\Psi \vdash \Pi x{:}\tau.K}$$

$$\frac{\Psi \vdash K \quad \Psi, t{::}K \vdash K'}{\Psi \vdash \Pi t{::}K.K'}$$

A.2 Well-formed (Functional) Types

$$\frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash \sigma :: \mathsf{type}}{\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}} \qquad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash \sigma :: K}{\Psi \vdash \lambda x{:}\tau.\sigma :: \Pi x{:}\tau.K}$$

$$\frac{\Psi \vdash \tau :: \Pi x{:}\sigma.K \quad \Psi \vdash M : \sigma}{\Psi \vdash \tau\, M :: K\{M/x\}} \qquad \frac{\forall i, j. \Psi \vdash A_i :: \mathsf{stype} \quad \Psi \vdash B_j :: \mathsf{stype} \quad \Psi \vdash A :: \mathsf{stype}}{\Psi \vdash \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\} :: \mathsf{type}}$$

$$\frac{\Psi \vdash K \quad \Psi, t{::}K \vdash \sigma :: K'}{\Psi \vdash \lambda t{::}K.\sigma :: \Pi t{::}K.K'} \qquad \frac{\Psi \vdash \tau :: \Pi t{::}K.K' \quad \Psi \vdash \sigma :: K}{\Psi \vdash \tau\, \sigma :: K'\{\sigma/t\}}$$

$$\frac{t{::}K \in \Psi \quad \Psi \vdash}{\Psi \vdash t :: K}$$

A.3 Well-formed Session Types

$$\frac{\Psi \vdash}{\Psi \vdash \mathbf{1} :: \mathsf{stype}} \qquad \frac{\Psi \vdash A :: \mathsf{stype}}{\Psi \vdash {!A} :: \mathsf{stype}} \qquad \frac{\Psi \vdash A :: \mathsf{stype} \quad \Psi \vdash B :: \mathsf{stype}}{\Psi \vdash A \multimap B :: \mathsf{stype}}$$

$$\frac{\Psi \vdash A :: \mathsf{stype} \quad \Psi \vdash B :: \mathsf{stype}}{\Psi \vdash A \otimes B :: \mathsf{stype}} \qquad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash A :: \mathsf{stype}}{\Psi \vdash \forall x{:}\tau.A :: \mathsf{stype}}$$

$$\frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash A :: \mathsf{stype}}{\Psi \vdash \exists x{:}\tau.A :: \mathsf{stype}} \qquad \frac{\forall i. \Psi \vdash A_i :: \mathsf{stype}}{\Psi \vdash \&\{\overline{l_i : A_i}\} :: \mathsf{stype}}$$

$$\frac{\forall i. \Psi \vdash A_i :: \mathsf{stype}}{\Psi \vdash \oplus\{\overline{l_i : A_i}\} :: \mathsf{stype}} \qquad \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash A :: K}{\Psi \vdash \lambda x{:}\tau.A :: \Pi x{:}\tau.K}$$

$$\frac{\Psi \vdash A :: \Pi x{:}\tau.K \quad \Psi \vdash M : \tau}{\Psi \vdash A\, M :: K\{M/x\}} \qquad \frac{\Psi \vdash A :: K \quad \Psi \vdash K = K'}{\Psi \vdash A :: K'}$$

$$\frac{\Psi, t{::}K \vdash A :: K'}{\Psi \vdash \lambda t{::}K.A :: \Pi t{::}K.K'} \qquad \frac{\Psi \vdash A :: \Pi t{::}K.K' \quad \Psi \vdash B :: K}{\Psi \vdash A\, B :: K'\{B/x\}} \qquad \frac{\Psi \vdash \quad t{::}K \in \Psi}{\Psi \vdash t :: K}$$

A.4 Typing for λ-Terms

$$(\Pi I)\ \frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \vdash M : \sigma}{\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma} \qquad (\Pi E)\ \frac{\Psi \vdash M : \Pi x{:}\tau.\sigma \quad \Psi \vdash N : \tau}{\Psi \vdash M\, N : \sigma\{N/x\}}$$

$$(\mathsf{var})\ \frac{\Psi \vdash \quad x{:}\tau \in \Psi}{\Psi \vdash x{:}\tau} \qquad (\{\}I)\ \frac{\forall i, j. \Psi \vdash A_i, B_j :: \mathsf{stype} \quad \Psi; \overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash P :: c{:}A}{\Psi \vdash \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} : \{\overline{u_j{:}B_j}; \overline{d_i : A_i} \vdash c{:}A\}}$$

$$(\mathsf{Conv})\ \frac{\Psi \vdash M : \tau \quad \Psi \vdash \tau = \sigma :: \mathsf{type}}{\Psi \vdash M : \sigma}$$

A.5 Typing for Processes

(∃R)

$$\frac{\Psi \vdash M{:}\tau \quad \Psi; \Gamma; \Delta \vdash P :: c{:}A\{M/x\}}{\Psi; \Gamma; \Delta \vdash c\langle M\rangle_{\exists x:\tau.A}.P :: c{:}\exists x{:}\tau.A}$$

(∃L)

$$\frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \; ; \Gamma; \Delta, c{:}A \vdash Q :: d{:}D}{\Psi \; ; \Gamma; \Delta, c{:}\exists x{:}\tau.A \vdash c(x{:}\tau).Q :: d{:}D}$$

(∀R)

$$\frac{\Psi \vdash \tau :: \mathsf{type} \quad \Psi, x{:}\tau \; ; \Gamma; \Delta \vdash P :: c{:}A}{\Psi; \Gamma; \Delta \vdash c(x{:}\tau).P :: c{:}\forall x{:}\tau.A}$$

(∀L)

$$\frac{\Psi \vdash M{:}\tau \quad \Psi; \Gamma; \Delta, c{:}A\{M/x\} \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta, c{:}\forall x{:}\tau.A \vdash c\langle M\rangle_{\forall x:\tau.A}.Q :: d{:}D}$$

(id)

$$\frac{\Psi; \Gamma \vdash \quad [\Psi \vdash A :: \mathsf{stype}]}{\Psi; \Gamma; d{:}A \vdash [d \leftrightarrow c] :: c{:}A}$$

($\mathbf{1}$R)

$$\frac{\Psi; \Gamma \vdash}{\Psi; \Gamma; \cdot \vdash \mathbf{0} :: c{:}\mathbf{1}}$$

($\mathbf{1}$L)

$$\frac{\Psi; \Gamma; \Delta \vdash P :: d{:}D}{\Psi; \Gamma; \Delta, c{:}\mathbf{1} \vdash P :: d{:}D}$$

(!R)

$$\frac{\Psi; \Gamma; \cdot \vdash P :: x{:}A}{\Psi; \Gamma; \cdot \vdash !c(x).P :: c{:}!A}$$

(!L)

$$\frac{\Psi; \Gamma, u{:}A; \Delta \vdash P :: d{:}D}{\Psi; \Gamma; \Delta, c{:}!A \vdash P\{c/u\} :: d{:}D}$$

($\textsc{copy}$)

$$\frac{\Psi; \Gamma, u{:}A; \Delta, x{:}A \vdash P :: d{:}D}{\Psi; \Gamma, u{:}A; \Delta \vdash (\boldsymbol{\nu} x)u\langle x\rangle.P :: d{:}D}$$

(⊗R)

$$\frac{\Psi; \Gamma; \Delta_1 \vdash P_1 :: x{:}A \quad \Psi; \Gamma; \Delta \vdash P_2 :: c{:}B}{\Psi; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)c\langle x\rangle.(P_1 \mid P_2) :: c{:}A \otimes B}$$

(⊗L)

$$\frac{\Psi; \Gamma; \Delta, x{:}A, c{:}B \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta, c{:}A \otimes B \vdash c(x).Q :: d{:}D}$$

(⊸R)

$$\frac{\Psi; \Gamma; \Delta, x{:}A \vdash P :: c{:}B}{\Psi; \Gamma; \Delta \vdash c(x).P :: c{:}A \multimap B}$$

(⊸L)

$$\frac{\Psi; \Gamma; \Delta_1 \vdash Q_1 :: x{:}A \quad \Psi; \Gamma; \Delta_2, c{:}B \vdash Q_2 :: d{:}D}{\Psi; \Gamma; \Delta_1, \Delta_2, c{:}A \multimap B \vdash (\boldsymbol{\nu} x)c\langle x\rangle.(Q_1 \mid Q_2) :: d{:}D}$$

(&R)

$$\frac{\Psi; \Gamma; \Delta \vdash P_1 :: c{:}A_1 \; \ldots \; \Psi; \Gamma; \Delta \vdash P_n :: c{:}A_n}{\Psi; \Gamma; \Delta \vdash c.\mathsf{case}(\overline{l_j \Rightarrow P_j}) :: c{:} \,\&\, \{\overline{l_j{:}A_j}\}}$$

(&L)

$$\frac{\Psi; \Gamma; \Delta, c{:}A_i \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta, c{:} \,\&\, \{\overline{l_j{:}A_j}\} \vdash c.l_i; Q :: d{:}D}$$

(⊕R)

$$\frac{\Psi; \Gamma; \Delta \vdash P :: c{:}A_i}{\Psi; \Gamma; \Delta \vdash c.l_i; P :: c{:} \oplus \{\overline{l_j{:}A_j}\}}$$

(⊕L)

$$\frac{\Psi; \Gamma; \Delta, c{:}A_1 \vdash Q_1 :: d{:}D \; \ldots \; \Psi; \Gamma; \Delta, c{:}A_n \vdash Q_n :: d{:}D}{\Psi; \Gamma; \Delta, c{:} \oplus \{\overline{l_j{:}A_j}\} \vdash c.\mathsf{case}(\overline{l_j \Rightarrow Q_j}) :: d{:}D}$$

($\textsc{cut}$)

$$\frac{\Psi; \Gamma; \Delta_1 \vdash P :: x{:}A \quad \Psi; \Gamma; \Delta_2, x{:}A \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: d{:}D}$$

($\textsc{cut}^!$)

$$\frac{\Psi; \Gamma; \cdot \vdash P :: x{:}A \quad \Psi; \Gamma, u{:}A; \Delta \vdash Q :: d{:}D}{\Psi; \Gamma; \Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid Q) :: d{:}D}$$

($\{\}E$)

$$\frac{\Delta' = \overline{d_i : B_i} \quad \overline{u_j{:}C_j} \subseteq \Gamma \quad \Psi \vdash M : \{\overline{u_j{:}C_j}; \overline{d_i{:}B_i} \vdash c{:}A\} \quad \Psi; \Gamma; \Delta, x{:}A \vdash Q :: z{:}C}{\Psi; \Gamma; \Delta', \Delta \vdash x \leftarrow M \leftarrow \overline{u_j}; \overline{y_i}; Q :: z{:}C}$$

(ConvR)

$$\frac{\Psi; \Gamma; \Delta \vdash P :: z{:}A \quad \Psi \vdash A = B :: \mathsf{stype}}{\Psi; \Gamma; \Delta \vdash P :: z{:}B}$$

(ConvL)

$$\frac{\Psi; \Gamma'; \Delta' \vdash P :: z{:}A \quad \Psi; \Gamma'; \Delta' = \Psi; \Gamma; \Delta}{\Psi; \Gamma; \Delta \vdash P :: z{:}A}$$

A.6 Definitional Equality for Kinds

$$\text{(KEqR)}\ \frac{\mathit{\Psi} \vdash K}{\mathit{\Psi} \vdash K = K} \qquad \text{(KEqS)}\ \frac{\mathit{\Psi} \vdash K_1 = K_2 \quad \mathit{\Psi} \vdash K_2 = K_3}{\mathit{\Psi} \vdash K_1 = K_3} \qquad \text{(KEqT)}\ \frac{\mathit{\Psi} \vdash K_2 = K_1}{\mathit{\Psi} \vdash K_1 = K_2}$$

$$\text{(KEq}\mathit{\Pi}\text{)}\ \frac{\mathit{\Psi} \vdash \tau = \sigma :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash K_1 = K_2}{\mathit{\Psi} \vdash \mathit{\Pi} x{:}\tau.K_1 = \mathit{\Pi} x{:}\sigma.K_2} \qquad \text{(KEqK}\mathit{\Pi}\text{)}\ \frac{\mathit{\Psi} \vdash K_1 = K_3 \quad \mathit{\Psi}, t{::}K_1 \vdash K_2 = K_4}{\mathit{\Psi} \vdash \mathit{\Pi} t :: K_1.K_2 = \mathit{\Pi} t :: K_3.K_4}$$

A.7 Definitional Equality for (Functional) Types

$$\text{(TEqR)}\ \frac{\mathit{\Psi} \vdash \tau :: \mathsf{type}}{\mathit{\Psi} \vdash \tau = \tau :: \mathsf{type}} \qquad \text{(TEqT)}\ \frac{\mathit{\Psi} \vdash \tau_1 = \tau_2 :: \mathsf{type} \quad \mathit{\Psi} \vdash \tau_2 = \tau_3 :: \mathsf{type}}{\mathit{\Psi} \vdash \tau_1 = \tau_3 :: \mathsf{type}}$$

$$\text{(TEqS)}\ \frac{\mathit{\Psi} \vdash \sigma = \tau :: \mathsf{type}}{\mathit{\Psi} \vdash \tau = \sigma :: \mathsf{type}} \qquad \text{(TEq}\mathit{\Pi}\text{)}\ \frac{\mathit{\Psi} \vdash \tau = \tau' :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash \sigma = \sigma' :: \mathsf{type}}{\mathit{\Psi} \vdash \mathit{\Pi} x{:}\tau.\sigma = \mathit{\Pi} x{:}\tau'.\sigma' :: \mathsf{type}}$$

$$\text{(TEq}\lambda\text{)}\ \frac{\mathit{\Psi} \vdash \tau = \tau' :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash \sigma = \sigma' :: K}{\mathit{\Psi} \vdash \lambda x{:}\tau.\sigma = \lambda x{:}\tau'.\sigma' :: \mathit{\Pi} x{:}\tau.K} \qquad \text{(TEqApp)}\ \frac{\mathit{\Psi} \vdash \tau = \sigma :: \mathit{\Pi} x{:}\tau'.K \quad \mathit{\Psi} \vdash M = N : \tau'}{\mathit{\Psi} \vdash \tau\, M = \sigma\, N :: K\{M/x\}}$$

$$\text{(TEq}\beta\text{)}\ \frac{\mathit{\Psi}, x{:}\tau \vdash \sigma :: K \quad \mathit{\Psi} \vdash M : \tau}{\mathit{\Psi} \vdash (\lambda x{:}\tau.\sigma)\, M = \sigma\{M/x\} :: K\{M/x\}} \qquad \text{(TEq}\eta\text{)}\ \frac{\mathit{\Psi} \vdash \sigma :: \mathit{\Pi} x{:}\tau.K \quad x \notin fv(\sigma)}{\mathit{\Psi} \vdash \lambda x{:}\tau.\sigma\, x = \sigma :: \mathit{\Pi} x{:}\tau.K}$$

$$\text{(TEq\{\})}\ \frac{\forall i, j. \quad \mathit{\Psi} \vdash A_i = B_i :: \mathsf{stype} \quad \mathit{\Psi} \vdash C_j = D_j :: \mathsf{stype} \quad \mathit{\Psi} \vdash A = B :: \mathsf{stype}}{\mathit{\Psi} \vdash \{\overline{u_j{:}C_j}; \overline{d_i{:}A_i} \vdash c{:}A\} = \{\overline{u_j{:}D_j}; \overline{d_i{:}B_i} \vdash c{:}B\} :: \mathsf{type}}$$

$$\text{(TEqT}\lambda\text{)}\ \frac{\mathit{\Psi} \vdash K_1 = K_2 \quad \mathit{\Psi}, t{::}K_1 \vdash \tau = \sigma :: K_3}{\mathit{\Psi} \vdash \lambda t{::}K_1.\tau = \lambda t{::}K_2.\sigma :: \mathit{\Pi} x{:}K_1.K_3} \qquad \text{(TEqTApp)}\ \frac{\mathit{\Psi} \vdash \tau_1 = \sigma_1 :: \mathit{\Pi} t{::}K_1.K_2 \quad \mathit{\Psi} \vdash \tau_2 = \sigma_2 : K_1}{\mathit{\Psi} \vdash \tau_1\, \tau_2 = \sigma_1\, \sigma_2 :: K_2\{\tau_2/t\}}$$

$$\text{(TEqT}\beta\text{)}\ \frac{\mathit{\Psi}, t{::}K \vdash \tau :: K' \quad \mathit{\Psi} \vdash \sigma :: K}{\mathit{\Psi} \vdash (\lambda t{::}K.\tau)\, \sigma = \tau\{\sigma/t\} :: K'\{\sigma/t\}} \qquad \text{(TEqConv)}\ \frac{\mathit{\Psi} \vdash \tau = \sigma :: K \quad \mathit{\Psi} \vdash K = K'}{\mathit{\Psi} \vdash \tau = \sigma :: K'}$$

A.8 Definitional Equality for Session Types

(STEqR)
$$\frac{\mathit{\Psi} \vdash A :: \mathsf{stype}}{\mathit{\Psi} \vdash A = A :: \mathsf{stype}}$$

(STEqS)
$$\frac{\mathit{\Psi} \vdash B = A :: \mathsf{stype}}{\mathit{\Psi} \vdash A = B :: \mathsf{stype}}$$

(STEqT)
$$\frac{\mathit{\Psi} \vdash A = B :: \mathsf{stype} \quad \mathit{\Psi} \vdash B = C :: \mathsf{stype}}{\mathit{\Psi} \vdash A = C :: s\mathsf{type}}$$

(STEq!)
$$\frac{\mathit{\Psi} \vdash A = B :: \mathsf{stype}}{\mathit{\Psi} \vdash\, !A = !B :: \mathsf{stype}}$$

(STEq$\multimap$)
$$\frac{\mathit{\Psi} \vdash A = C :: \mathsf{stype} \quad \mathit{\Psi} \vdash B = D :: \mathsf{stype}}{\mathit{\Psi} \vdash A \multimap B = C \multimap D :: \mathsf{stype}}$$

(STEq$\otimes$)
$$\frac{\mathit{\Psi} \vdash A = C :: \mathsf{stype} \quad \mathit{\Psi} \vdash B = D :: \mathsf{stype}}{\mathit{\Psi} \vdash A \otimes B = C \otimes D :: \mathsf{stype}}$$

(STEq$\forall$)
$$\frac{\mathit{\Psi} \vdash \tau = \tau' :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash A = B :: \mathsf{stype}}{\mathit{\Psi} \vdash \forall x{:}\tau.A = \forall x{:}\tau'.B :: \mathsf{stype}}$$

(STEq$\exists$)
$$\frac{\mathit{\Psi} \vdash \tau = \tau' :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash A = B :: \mathsf{stype}}{\mathit{\Psi} \vdash \exists x{:}\tau.A = \exists x{:}\tau'.B :: \mathsf{stype}}$$

(STEq$\&$)
$$\frac{\forall i.\mathit{\Psi} \vdash A_i = B_i :: \mathsf{stype}}{\mathit{\Psi} \vdash \&\{\overline{l_i{:}A_i}\} = \&\{\overline{l_i{:}B_i}\} :: \mathsf{stype}}$$

(STEq$\oplus$)
$$\frac{\forall i.\mathit{\Psi} \vdash A_i = B_i :: \mathsf{stype}}{\mathit{\Psi} \vdash \oplus\{\overline{l_i{:}A_i}\} = \oplus\{\overline{l_i{:}B_i}\} :: \mathsf{stype}}$$

(STEqλ)
$$\frac{\mathit{\Psi} \vdash \tau = \tau' :: \mathsf{type} \quad \mathit{\Psi}, x{:}\tau \vdash A = B :: K}{\mathit{\Psi} \vdash \lambda x{:}\tau.A = \lambda x{:}\tau'.B :: \mathit{\Pi} x{:}\tau.K}$$

(STEqApp)
$$\frac{\mathit{\Psi} \vdash A = B :: \mathit{\Pi} x{:}\tau.K \quad \mathit{\Psi} \vdash M = N : \tau}{\mathit{\Psi} \vdash A\,M = B\,N :: K\{M/x\}}$$

(STEqβ)
$$\frac{\mathit{\Psi}, x{:}\tau \vdash A :: K \quad \mathit{\Psi} \vdash M : \tau}{\mathit{\Psi} \vdash (\lambda x{:}\tau.A)\,M = A\{M/x\} :: K\{M/x\}}$$

(STEqη)
$$\frac{\mathit{\Psi} \vdash A :: \mathit{\Pi} x{:}\tau.K \quad x \notin fv(A)}{\mathit{\Psi} \vdash \lambda x{:}\tau.A\,x = A :: \mathit{\Pi} x{:}\tau.K}$$

(STEqTλ)
$$\frac{\mathit{\Psi} \vdash K_1 = K_2 \quad \mathit{\Psi}, t{::}K_1 \vdash A = B :: K_3}{\mathit{\Psi} \vdash \lambda t{::}K_1.A = \lambda t{::}K_2.B :: \mathit{\Pi} x{:}K_1.K_3}$$

(STEqTApp)
$$\frac{\mathit{\Psi} \vdash A = C :: \mathit{\Pi} t{::}K_1.K_2 \quad \mathit{\Psi} \vdash B = D : K_1}{\mathit{\Psi} \vdash A\,B = C\,D :: K_2\{B/t\}}$$

(STEqTβ)
$$\frac{\mathit{\Psi}, t{::}K \vdash A :: K' \quad \mathit{\Psi} \vdash B :: K}{\mathit{\Psi} \vdash (\lambda t{::}K.A)\,B = A\{B/t\} :: K'\{B/t\}}$$

(STEqConv)
$$\frac{\mathit{\Psi} \vdash A = B :: K \quad \mathit{\Psi} \vdash K = K'}{\mathit{\Psi} \vdash A = B :: K'}$$

A.9 Definitional Equality for $\boldsymbol{\lambda}$-Terms

(TMEqR)
$$\frac{\Psi \vdash M : \tau}{\Psi \vdash M = M : \tau}$$

(TMEqS)
$$\frac{\Psi \vdash N = M : \tau}{\Psi \vdash M = N : \tau}$$

(TMEqT)
$$\frac{\Psi \vdash M = N' : \tau \quad \Psi \vdash N' = N : \tau}{\Psi \vdash M = N : \tau}$$

(TMEqVar)
$$\frac{\Psi \vdash \quad x{:}\tau \in \Psi}{\Psi \vdash x = x : \tau}$$

(TMEqλ)
$$\frac{\begin{array}{c}\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma \quad \Psi \vdash \lambda x{:}\tau'.N : \Pi x{:}\tau'.\sigma' \\ \Psi \vdash \Pi x{:}\tau.\sigma = \Pi x{:}\tau'.\sigma' :: \mathsf{type} \quad \Psi, x{:}\tau \vdash M = N : \sigma\end{array}}{\Psi \vdash \lambda x{:}\tau.M = \lambda x{:}\tau'.N : \Pi x{:}\tau.\sigma}$$

(TMEqApp)
$$\frac{\Psi \vdash M = M' : \Pi x{:}\tau.\sigma \quad \Psi \vdash N = N' : \tau}{\Psi \vdash M\,N = M'\,N' : \sigma\{N/x\}}$$

(TMEqβ)
$$\frac{[\Psi \vdash \tau :: \mathsf{type}] \quad \Psi, x{:}\tau \vdash M : \sigma \quad \Psi \vdash N : \tau}{\Psi \vdash (\lambda x{:}\tau.M)\,N = M\{N/x\} : \sigma\{N/x\}}$$

(TMEqη)
$$\frac{\Psi \vdash M : \Pi x{:}\tau.\sigma \quad x \notin fv(M)}{\Psi \vdash \lambda x{:}\tau.M\,x = M : \Pi x{:}\tau.\sigma}$$

(TMEq$\{\}\eta$)
$$\frac{\Psi \vdash M : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}}{\Psi \vdash \{c \leftarrow (y \leftarrow M; \overline{u_j}; \overline{d_i}; [y \leftrightarrow c]) \leftarrow \overline{u_j}; \overline{d_i}\} = M : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}}$$

(TMEq$\{\}$)
$$\frac{[\forall i, j.\Psi \vdash B_j :: \mathsf{stype} \quad \Psi \vdash A_i :: \mathsf{stype}] \quad \Psi; \overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash P = Q :: c{:}A}{\Psi \vdash \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} = \{c \leftarrow Q \leftarrow \overline{u_j}; \overline{d_i}\} : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}}$$

(TMEqConv)
$$\frac{\Psi \vdash M = N : \tau \quad \Psi \vdash \tau = \sigma :: \mathsf{type}}{\Psi \vdash M = N : \sigma}$$

A.10 Definitional Equality for Processes

$$\text{(PEqRefl)}\ \frac{\Psi;\Gamma;\Delta \vdash P :: z{:}A}{\Psi;\Gamma;\Delta \vdash P = P :: z{:}A} \qquad \text{(PEqS)}\ \frac{\Psi;\Gamma;\Delta \vdash Q = P :: z{:}A}{\Psi;\Gamma;\Delta \vdash P = Q :: z{:}A}$$

$$\text{(PEqT)}\ \frac{\Psi;\Gamma;\Delta \vdash P = Q :: z{:}A \quad \Psi;\Gamma;\Delta \vdash Q = R :: z{:}A}{\Psi;\Gamma;\Delta \vdash P = R :: z{:}A}$$

$$\frac{\Psi;\Gamma;\Delta \vdash P :: z{:}A \quad P \to Q \quad \Psi;\Gamma;\Delta \vdash Q :: z{:}A}{\Psi;\Gamma;\Delta \vdash P = Q :: z{:}A}$$

$$\text{(PEq}\forall\eta\text{)}\ \frac{}{\Psi;\Gamma;d{:}\forall x{:}\tau.A \vdash c(x).d\langle x\rangle.[d \leftrightarrow c] = [d \leftrightarrow c] :: c{:}\forall x{:}\tau.A}$$

$$\text{(PEqCC}\forall\text{)}\ \frac{\Psi;\Gamma;\Delta \vdash P :: d{:}B \quad \Psi, x{:}\tau;\Gamma;\Delta', d{:}B \vdash Q :: c{:}A}{\Psi;\Gamma;\Delta,\Delta' \vdash (\boldsymbol{\nu} d)(P \mid c(x).Q) = c(x).(\boldsymbol{\nu} d)(P \mid Q) :: c{:}\forall x{:}\tau.A}$$

$$\text{(PEq}\forall\text{R)}\ \frac{\begin{array}{c}\Psi;\Gamma;\Delta \vdash z(x{:}\tau).P :: z{:}\forall x{:}\tau.A \quad \Psi;\Gamma;\Delta \vdash z(x{:}\tau').Q :: z{:}\forall x{:}\tau'.B \\ \Psi \vdash \forall x{:}\tau.A = \forall x{:}\tau'.B :: \mathsf{stype} \quad \Psi, x{:}\tau;\Gamma;\Delta \vdash P = Q :: z{:}A\end{array}}{\Psi;\Gamma;\Delta \vdash z(x{:}\tau).P = z(x{:}\tau').Q :: z{:}\forall x{:}\tau.A}$$

(Other congruence, η and CC rules)

B Type Soundness

We use $\Psi \vdash \mathcal{J}$ to signify any of the judgments $\Psi \vdash K$, $\Psi \vdash A :: K$, $\Psi \vdash \tau :: K$ and respective definitional equality judgments. We use $\Psi;\Gamma;\Delta \vdash \mathcal{J}$ in a similar fashion.

Lemma B.1 (Subderivation Properties).

1. *Every derivation of $\Psi \vdash \mathcal{J}$ has a proof of $\Psi \vdash$ as a sub-proof.*
2. *Every derivation of $\Psi, x{:}\tau \vdash$ has a proof of $\Psi \vdash \tau :: \mathsf{type}$ as a sub-proof.*
3. *Every derivation of $\Psi, t{::}K \vdash$ has a proof of $\Psi \vdash K$ as a sub-proof.*
4. *Every derivation of $\Psi, x{:}K \vdash$ has a proof of $\Psi \vdash K$ as a sub-proof.*
5. *If $\Psi \vdash \tau :: K$ or $\Psi \vdash A :: K$ then $\Psi \vdash K$*
6. *If $\Psi \vdash M : \tau$ then $\Psi \vdash \tau :: \mathsf{type}$*
7. *If $\Psi;\Gamma;\Delta \vdash P :: z{:}A$ then $\Psi \vdash A :: \mathsf{stype}$*

Proof. By induction on the given derivation.

Case: Kind well-formedness
Straightforward by induction.
Case: Functional type well-formedness
Straightforward by induction.
Case: Session type well-formedness
Straightforward by induction.

Case: Typing for terms
Straightforward by induction. Base-case is immediate.
Case: Typing for processes
Straightforward by induction. Base-cases are immediate.
Case: Kind equivalence
Straightforward, base case is reflexivity (from i.h. for well-formedness).
Case: Type Equivalence
As above.
Case: Session type equivalence
As above.

Lemma B.2 (Weakening). *If $\Psi \vdash$ and $\Psi' \vdash$ and $\Psi \subseteq \Psi'$ then:*

1. $\Psi \vdash \mathcal{J}$ *implies* $\Psi' \vdash \mathcal{J}$
2. $\Psi; \Gamma; \Delta \vdash \mathcal{J}$ *implies* $\Psi'; \Gamma; \Delta \vdash \mathcal{J}$

Proof. Straightforward induction on the given derivation.

Lemma 2.1 (Substitution). *Let $\Psi \vdash M : \tau$:*

1. *If $\Psi, x{:}\tau, \Psi' \vdash \mathcal{J}$ then $\Psi, \Psi'\{M/x\} \vdash \mathcal{J}\{M/x\}$;*
2. *If $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash \mathcal{J}$ then $\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash \mathcal{J}\{M/x\}$*

Proof. By induction on the second given derivation. We show only some illustrative cases.

Case: TypeAppWF

$\Psi, x{:}\tau, \Psi' \vdash \tau' :: \Pi y{:}\sigma.K$ and $\Psi, x{:}\tau, \Psi' \vdash M' : \sigma$	by inversion
$\Psi, \Psi' \vdash \tau'\{M/x\} :: \Pi y{:}\sigma\{M/x\}.K\{M/x\}$	by i.h.
$\Psi, \Psi' \vdash M'\{M/x\} : \sigma\{M/x\}$	by i.h.
$\Psi, \Psi' \vdash \tau'\{M/x\}\, M'\{M/x\} : K\{M'/y\}\{M/x\}$	by TypeAppWF

Case: KindConv

$\Psi, x{:}\tau, \Psi' \vdash \tau :: K$ and $\Psi, x{:}\tau, \Psi' \vdash K = K'$	by inversion
$\Psi, \Psi' \vdash \tau\{M/x\} :: K\{M/x\}$	by i.h.
$\Psi, \Psi' \vdash K\{M/x\} = K'\{M/x\}$	by i.h.
$\Psi, \Psi' \vdash \tau\{M/x\} :: K'\{M/x\}$	by KindConv

Case: Var

Subcase: $x = y$

$\Psi \vdash M : \tau$	by assumption
$\Psi, \Psi'\{M/x\} \vdash M : \tau$	by weakening

Subcase: $x \neq y$

$\Psi, x{:}\tau, \Psi', y{:}\tau' \vdash y{:}\tau'$	by weakening and Var

Case: TEqβ

$\Psi, x{:}\tau, \Psi', y{:}\tau' \vdash \sigma :: K$ and $\Psi, x{:}\tau, \Psi' \vdash M' : \tau'$ by inversion
$\Psi, \Psi'\{M/x\}, y{:}\tau'\{M/x\} \vdash \sigma\{M/x\} :: K\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash M'\{M/x\} : \tau'\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash (\lambda y{:}\tau'\{M/x\}.\sigma\{M/x\})\, M'\{M/x\} = \sigma\{M/x\}\{M'\{M/x\}/y\} :: K\{M'/x\}\{M\{M/x\}/y\}$
by $\mathsf{TEq}\beta$

Case: $\mathsf{TEq}\eta$

$\Psi, x{:}\tau, \Psi' \vdash \sigma :: \Pi y{:}\tau'.K$ and $y \notin fv(\sigma)$ by inversion
$\Psi, \Psi'\{M/x\} \vdash \sigma\{M/x\} :: \Pi y{:}\tau'\{M/x\}.K\{M/x\}$ by i.h
$\Psi, \Psi'\{M/\} \vdash \lambda y{:}\tau'\{M/x\}.(\sigma\{M/x\}\, y) = \sigma\{M/x\} :: \Pi y{:}\tau'\{M/x\}.K\{M/x\}$ by $\mathsf{TEq}\eta$

Case: PEqRed

$\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash P :: z{:}A,\ P \to^* Q$ and $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash Q :: z{:}A$ by inversion
$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} :: z{:}A\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash Q\{M/x\} :: z{:}A\{M/x\}$ by i.h.
$P\{M/x\} \to^* Q\{M/x\}$ by compatibility of reduction with substitution
$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = Q\{M/x\} :: z{:}A\{M/x\}$ by PEqRed

Lemma B.3 (Type Substitution).

1. *If $\Psi \vdash \tau :: K$ and $\Psi, t{::}K, \Psi' \vdash \mathcal{J}$ then $\Psi, \Psi'\{\tau/t\} \vdash \mathcal{J}\{\tau/t\}$;*
2. *If $\Psi \vdash \tau :: K$ and $\Psi, t{::}K, \Psi'; \Gamma; \Delta \vdash \mathcal{J}$ then $\Psi, \Psi'\{\tau/t\}; \Gamma\{\tau/t\}; \Delta\{\tau/t\} \vdash \mathcal{J}\{\tau/t\}$*
3. *If $\Psi \vdash A :: K$ and $\Psi, t{::}K, \Psi' \vdash \mathcal{J}$ then $\Psi, \Psi'\{A/t\} \vdash \mathcal{J}\{A/t\}$;*
4. *If $\Psi \vdash A :: K$ and $\Psi, t{::}K, \Psi'; \Gamma; \Delta \vdash \mathcal{J}$ then $\Psi, \Psi'\{A/t\}; \Gamma\{A/t\}; \Delta\{A/t\}$*

Lemma B.4 (Context Conversion).
Let $\Psi, x{:}\tau \vdash$ and $\Psi \vdash \tau' :: K$. If $\Psi, x{:}\tau \vdash \mathcal{J}$ and $\Psi \vdash \tau = \tau' :: K$ then $\Psi, x{:}\tau' \vdash \mathcal{J}$.

Proof. Straightforward from the properties above.

$\Psi, x{:}\tau' \vdash x{:}\tau'$ by variable rule
$\Psi \vdash \tau' = \tau :: K$ by symmetry
$\Psi, x{:}\tau' \vdash x{:}\tau$ by conversion
$\Psi, x'{:}\tau \vdash \alpha\{x'/x\}$ renaming assumption
$\Psi, x{:}\tau', x'{:}\tau \vdash \alpha\{x'/x\}$ by weakening
$\Psi, x{:}\tau' \vdash \alpha\{x'/x\}\{x/x'\}$ by substitution
$\Psi, x{:}\tau' \vdash \alpha$ by definition

Lemma B.5 (Context Conversion – Processes). *Let $\Psi, x{:}\tau; \Delta \vdash$, $\Psi, x{:}\tau; \Gamma \vdash$ and $\Psi \vdash \tau :: \mathsf{type}$. If $\Psi, x{:}\tau; \Gamma; \Delta \vdash \mathcal{J}$ and $\Psi \vdash \tau = \tau' :: \mathsf{type}$ then $\Psi, x{:}\tau'; \Gamma; \Delta \vdash \mathcal{J}$*

Proof. Straightforward by Lemma B.4.

Lemma B.6 (Context Conversion – Types). *Let $\Psi, t{::}K \vdash$ and $\Psi \vdash K'$. If $\Psi, t{::}K \vdash \mathcal{J}$ and $\Psi \vdash K = K'$ then $\Psi, t{::}K' \vdash \mathcal{J}$*

Proof. Identical to Lemma B.4

Lemma B.7 (Functionality of Typing).
Assume $\Psi \vdash M = N : \tau$, $\Psi \vdash M : \tau$ *and* $\Psi \vdash N : \tau$:

1. *If* $\Psi, x{:}\tau, \Psi' \vdash M' : \tau'$ *then* $\Psi, \Psi'\{M/x\} \vdash M'\{M/x\} = M'\{N/x\} : \tau'\{M/x\}$
2. *If* $\Psi, x{:}\tau, \Psi' \vdash \tau' :: K$ *then* $\Psi, \Psi'\{M/x\} \vdash \tau'\{M/x\} = \tau'\{N/x\} :: K\{M/x\}$
3. *If* $\Psi, x{:}\tau, \Psi' \vdash A :: K$ *then* $\Psi, \Psi'\{M/x\} \vdash A\{M/x\} = A\{N/x\} :: K\{M/x\}$
4. *If* $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash P :: z{:}A$ *then* $\Psi; \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = P\{N/x\} :: z{:}A\{N/x\}$
5. *If* $\Psi, x{:}\tau, \Psi' \vdash K$ *then* $\Psi, \Psi'\{M/x\} \vdash K\{M/x\} = K\{N/x\}$

Proof. By induction on the given typing derivation.

Case: $\Psi, x{:}\tau, \Psi' \vdash x{:}\tau$ by variable rule

$\Psi \vdash M = N : \tau$ by assumption
$\Psi, \Psi'\{M/x\} \vdash M = N : \tau$ by weakening

Case: $\Psi, x{:}\tau, \Psi' \vdash y{:}\sigma$ with $y{:}\sigma \in \Psi$ or Ψ'

$y : \sigma \in \Psi$ or $y{:}\sigma\{M/x\} \in \Psi'\{M/x\}$ by definition
$\Psi, \Psi'\{M/x\} \vdash y = y : \sigma\{M/x\}$ by reflexivity

Case: $\Psi, x{:}\tau, \Psi' \vdash M_0\, N_0 : \sigma_0\{N_0/y\}$ from ΠE

$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \Pi y{:}\sigma_1\{M/x\}.\sigma_0\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash N_0\{M/x\} = N_0\{N/x\} : \sigma_1\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\}\, N_0\{M/x\} = M_0\{N/x\}\, N_0\{N/x\} : (\sigma_0\{M/x\})\{(N_0\{M/x\})/y\}$
by $\mathsf{TMEqApp}$ rule

Case: $\Psi, x{:}\tau, \Psi' \vdash \lambda y{:}\tau_0.M_0 : \Pi y{:}\tau_0.\tau_1$ by ΠI rule

$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: \mathsf{type}$ by i.h.
$\Psi, \Psi'\{M/x\}, y{:}\tau_0\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \tau_1\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} :: \mathsf{type}$ by substitution lemma
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{M/x\} :: \mathsf{type}$ by reflexivity
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{N/x\} = \tau_0\{M/x\} :: \mathsf{type}$ by symmetry
$\Psi, \Psi'\{M/x\} \vdash \lambda y{:}\tau_0\{M/x\}.M_0\{M/x\} = \lambda y{:}\tau_0\{N/x\}.M_0\{N/x\} : \Pi y{:}\tau_0\{M/x\}.\tau_1\{M/x\}$
by $\mathsf{TMEq}\lambda$ rule

Case: $\Psi, x{:}\tau, \Psi' \vdash \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} : \{\Gamma; \Delta \vdash c{:}A\}$ by $\{\}I$

$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = P\{N/x\} :: c{:}A\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \overline{A_j\{M/x\}} :: \mathsf{stype}$ by substitution lemma
$\Psi, \Psi'\{M/x\} \vdash \overline{B_i\{M/x\}} :: \mathsf{stype}$ by substitution lemma
Conclude by $\mathsf{TMEq}\{\}$ rule

Case: $\Psi, x{:}\tau, \Psi' \vdash M_0 : \tau_0$ by conversion rule

$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \tau_0'\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0'\{M/x\} :: \mathsf{type}$ by substitution lemma
$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \tau_0\{M/x\}$ by conversion rule

Case: $\Psi, x{:}\tau, \Psi' \vdash \Pi y{:}\tau_0.\tau_1 :: \mathsf{type}$ by Π formation rule

$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} :: \mathsf{type}$ by substitution
$\Psi, \Psi'\{M/x\}, y{:}\tau_0\{M/x\} \vdash \tau_1\{M/x\} = \tau_1\{N/x\} :: \mathsf{type}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: \mathsf{type}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \Pi y{:}\tau_0\{M/x\}.\tau_1\{M/x\} = \Pi y{:}\tau_0\{N/x\}.\tau_1\{N/x\} :: \mathsf{type}$
by Π formation rule

Case: $\Psi, x{:}\tau, \Psi' \vdash \lambda y{:}\tau_0.\sigma :: \Pi y{:}\tau_0.K_0$ by λ formation rule

$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: \mathsf{type}$ by i.h.
$\Psi, \Psi'\{M/x\}, y{:}\tau_0\{M/x\} \vdash \sigma\{M/x\} = \sigma\{N/x\} :: K_0\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \lambda y{:}\tau_0\{M/x\}.\sigma\{M/x\} = \lambda y{:}\tau_0\{N/x\}.\sigma\{N/x\} :: \Pi y{:}\tau_0\{M/x\}.K_0\{M/x\}$
by λ formation rule

Case: $\Psi, x{:}\tau, \Psi' \vdash \tau_0\, M_0 :: K_0\{M/y\}$ by type application formation rule

$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: \Pi y{:}\tau_1\{M/x\}.K_0\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \tau_1\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\}\, M_0\{M/x\} = \tau_0\{M/x\}\, M_0\{M/x\} :: K_0\{M_0/y\}\{M/x\}$
by type app. formation rule and def. of substitution

Case: $\{\}$ formation rule

Straightforward by i.h.

Case: $\Psi, x{:}\tau, \Psi' \vdash \tau_0 :: K_0$ by conversion rule

$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: K_1\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash K_1\{M/x\} = K_0\{M/x\}$ by substitution lemma
$\Psi, \Psi'\{M/x\} \vdash \tau_0\{M/x\} = \tau_0\{N/x\} :: K_0\{M/x\}$ by conversion

Case: $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash c\langle M_0\rangle.P_0 :: c{:}\exists y{:}\tau_0.A_0$ by $\exists\mathsf{R}$

$\Psi, \Psi'\{M/x\} \vdash M_0\{M/x\} = M_0\{N/x\} : \tau_0\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P_0\{M/x\} = P_0\{N/x\} :: c{:}A_0\{M_0/y\}\{M/x\}$
by i.h.
Conclude by $\mathsf{PEq}\exists\mathsf{R}$

Case: $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta, y{:}\exists w{:}\sigma.A \vdash y(w{:}\sigma).P_0 :: z{:}C$ by $\exists\mathsf{L}$

$\Psi, \Psi'\{M/x\}, w{:}\sigma\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\}, y{:}A\{M/x\} \vdash P_0\{M/x\} = P_0\{N/x\} :: z{:}C\{M/x\}$
by i.h.
$\Psi, \Psi'\{M/x\} \vdash \sigma\{M/x\} :: \mathsf{type}$ by substitution lemma
Conclude by $\mathsf{PEq}\exists\mathsf{L}$

Case: $\Psi, x{:}\tau, \Psi'; \Gamma; \Delta \vdash P :: z{:}B$ by ConvR

$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = P\{N/x\} :: z{:}A\{M/x\}$ by i.h.
$\Psi, \Psi'\{M/x\} \vdash A\{M/x\} = B\{M/x\} :: \mathsf{stype}$ by substitution lemma
$\Psi, \Psi'\{M/x\}; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = P\{N/x\} :: z{:}B\{M/x\}$
by conversion

Remaining cases follow similar patterns, relying on the inductive hypothesis and the substitution lemmata.

We omit the analogue functionality property for type substitution.

Lemma B.8 (Inversion for Products).

1. *If* $\Psi \vdash \Pi x{:}\tau.\sigma :: K$ *then* $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$
2. *If* $\Psi \vdash \Pi x{:}\tau.K$ *then* $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash K$

Proof. (1) follows straightforwardly by induction on the given derivation. (2) is immediate by inversion.

Lemma B.9 (Inversion for $\forall\exists$).

1. *If* $\Psi \vdash \forall x{:}\tau.A :: K$ *then* $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash A :: \mathsf{stype}$
2. *If* $\Psi \vdash \exists x{:}\tau.A : K$ *then* $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash A :: \mathsf{stype}$

Proof. Straightforwardly by induction on the given derivation.

Lemma 2.3 (Validity for Equality).

1. *If* $\Psi \vdash M = N : \tau$ *then* $\Psi \vdash M : \tau$*,* $\Psi \vdash N : \tau$ *and* $\Psi \vdash \tau :: \mathsf{type}$
2. *If* $\Psi \vdash \tau = \sigma :: K$ *then* $\Psi \vdash \tau :: K$*,* $\Psi \vdash \sigma :: K$ *and* $\Psi \vdash K$
3. *If* $\Psi \vdash A = B :: K$ *then* $\Psi \vdash A :: K$*,* $\Psi \vdash B :: K$ *and* $\Psi \vdash K$
4. *If* $\Psi \vdash K = K'$ *then* $\Psi \vdash K$ *and* $\Psi \vdash K'$
5. *If* $\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$ *then* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$*,* $\Psi; \Gamma; \Delta \vdash Q :: z{:}A$ *and* $\Psi \vdash A :: \mathsf{stype}$

Proof. By simultaneous induction on the given derivation.

Case: TMEqR

$\Psi \vdash M : \tau$ by inversion
$\Psi \vdash \tau :: \mathsf{type}$ by subderivation lemma

Case: TMEqS and TMEqT

Immediate by i.h.

Case: $\mathsf{TMEq}\lambda$

$\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma$, $\Psi \vdash \lambda x{:}\tau'.N : \Pi x{:}\tau'.\sigma'$, $\Psi \vdash \Pi x{:}\tau.\sigma = \Pi x{:}\tau'.\sigma' :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash M = N : \sigma$ by inversion

$\Psi, x{:}\tau \vdash M : \sigma$, $\Psi, x{:}\tau \vdash N : \sigma$ and $\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$ by i.h.

$\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$, $\Psi \vdash \Pi x{:}\tau'.\sigma' :: \mathsf{type}$ and $\Psi \vdash \mathsf{type}$ by i.h.

$\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma$ by (ΠI)

$\Psi \vdash \lambda x{:}\tau'.N : \Pi x{:}\tau'.\sigma'$ by (ΠI)

$\Psi \vdash \lambda x{:}\tau'.N : \Pi x{:}\tau.\sigma$ by conversion (and symmetry)

Case: $\mathsf{TMEqApp}$

$\Psi \vdash M = M' : \Pi x{:}\tau.\sigma$ and $\Psi \vdash N = N' : \tau$ by inversion

$\Psi \vdash M : \Pi x{:}\tau.\sigma$, $\Psi \vdash M' : \Pi x{:}\tau.\sigma$ and $\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$ by i.h.

$\Psi \vdash N : \tau$, $\Psi \vdash N' : \tau$ and $\Psi \vdash \tau :: \mathsf{type}$ by i.h.

$\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$ by inversion for products

$\Psi \vdash \sigma\{N/x\} :: \mathsf{type}$ by substitution

$\Psi \vdash M\, N : \sigma\{N/x\}$ by (ΠE)

$\Psi \vdash M'\, N' : \sigma\{N'/x\}$ by (ΠE)

$\Psi \vdash \sigma\{N/x\} = \sigma\{N'/x\} :: \mathsf{type}$ by functionality

$\Psi \vdash M'\, N' : \sigma\{N/x\}$ by conversion (and symmetry)

Case: $\mathsf{TMEq}\beta$

$\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma$, $\Psi \vdash N : \tau$ and $\Psi, {:}\tau \vdash M : \sigma$ by inversion

$\Psi \vdash (\lambda x{:}\tau.M)\, N : \sigma\{N/x\}$ by (ΠE)

$\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$ by subderivation lemma

$\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$ by inversion for products

$\Psi \vdash \sigma\{N/x\} :: \mathsf{type}$ by substitution

$\Psi \vdash M\{N/x\} : \sigma\{N/x\}$ by substitution

Case: $\mathsf{TMEq}\eta$

$\Psi \vdash M : \Pi x{:}\tau.\sigma$ by inversion

$\Psi \vdash \lambda x{:}\tau.(M\, x) : \Pi x{:}\tau, \sigma$ by (ΠE), (var) and (ΠI)

$\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$ by subderivation lemma

Case: $\mathsf{TMEq}\{\}$

$\Psi; \Gamma; \Delta \vdash P = Q :: c{:}A$ by inversion

$\Psi; \Gamma; \Delta \vdash P :: c{:}A$, $\Psi; \Gamma; \Delta \vdash Q :: c{:}A$ and $\Psi \vdash A :: \mathsf{stype}$ by i.h.

$\Psi \vdash \{c \leftarrow P \leftarrow \dots\} : \{\Gamma; \Delta \vdash c{:}A\}$ by $\{\}I$

$\Psi \vdash \{c \leftarrow Q \leftarrow \dots\} : \{\Gamma; \Delta \vdash c{:}A\}$ by $\{\}I$

$\Psi; \Gamma \vdash$ and $\Psi; \Delta \vdash$ by subderivation lemma

$\Psi \vdash \{\Gamma; \Delta \vdash c{:}A\}$ by $\{\}$ well-formedness

Case: $\mathsf{TMEq}\{\}\eta$

$\Psi \vdash M : \{\Gamma; \Delta \vdash c{:}A\}$ by inversion

$\Psi \vdash \{\Gamma; \Delta \vdash c{:}A\}$ by subderivation lemma

Typing follows straightforwardly

Case: TEqR

Straightforward by subderivation lemma.

Case: TEqS and TEqT

Straightforward by i.h.

Case: $\mathsf{TEq}\Pi$

$\Psi \vdash \tau = \tau' :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \sigma = \sigma' :: \mathsf{type}$	by inversion
$\Psi \vdash \tau :: \mathsf{type}$, $\Psi \vdash \tau' :: \mathsf{type}$ and $\Psi \vdash \mathsf{type}$	by i.h.
$\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$, $\Psi, x{:}\tau \vdash \sigma' :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \mathsf{type}$	by i.h.
$\Psi \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$	Π rule
$\Psi, x{:}\tau' \vdash \sigma' :: \mathsf{type}$	by context conversion
$\Psi \vdash \Pi x{:}\tau.\sigma' :: \mathsf{type}$	by Π rule

Case: $\mathsf{TEq}\lambda$

$\Psi \vdash \tau = \tau' :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \sigma = \sigma' :: K$	by inversion
$\Psi \vdash \tau :: \mathsf{type}$, $\Psi \vdash \tau' :: \mathsf{type}$ and $\Psi \vdash \mathsf{type}$	by i.h.
$\Psi, x{:}\tau \vdash \sigma :: K$, $\Psi, x{:}\tau \vdash \sigma' :: K$ and $\Psi, x{:}\tau \vdash K$	by i.h.
$\Psi \vdash \lambda x{:}\tau.\sigma :: \Pi x{:}\tau.K$	by λ rule
$\Psi, x{:}\tau' \vdash \sigma' :: K$	by context conversion
$\Psi \lambda x{:}\tau'.\sigma' :: \Pi x{:}\tau'.K$	by λ rule
$\Psi \vdash \lambda x{:}\tau'.\sigma' :: \Pi x{:}\tau.K$	by conversion
$\Psi \vdash \Pi x{:}\tau.K$	by Π well-formedness rule

Case: TEqApp

$\Psi \vdash \tau = \sigma :: \Pi x{:}\tau'.K$ and $\Psi \vdash M = N : \tau'$	by inversion
$\Psi \vdash \tau :: \Pi x{:}\tau'.K$, $\Psi \vdash \sigma :: \Pi x{:}\tau'.K$ and $\Psi \vdash \Pi x{:}\tau'.K$	by i.h.
$\Psi \vdash M : \tau'$, $\Psi \vdash N : \tau'$ and $\Psi \vdash \tau' :: \mathsf{type}$	by i.h.
$\Psi \vdash \tau\, M : K\{M/x\}$	by app. wf rule
$\Psi \vdash \sigma\, N : K\{N/x\}$	by app. wf rule
$\Psi, x{:}\tau' \vdash K$	by inversion for products
$\Psi \vdash K\{M/x\} = K\{N/x\}$	by functionality
$\Psi \vdash \sigma\, N : K\{M/x\}$	by conversion
$\Psi \vdash K\{M/x\}$	by substitution

Case: $\mathsf{TEq}\beta$

$\Psi, x{:}\tau \vdash \sigma :: K$ and $\Psi \vdash M : \tau$	by inversion
$\Psi \vdash \lambda x{:}\tau.\sigma :: \Pi x{:}\tau.K$	by Π rule
$\Psi \vdash (\lambda x{:}\tau.\sigma)\, M :: K\{M/x\}$	by app. rule
$\Psi \vdash \sigma\{M/x\} :: K\{M/x\}$	by substitution
$\Psi, x{:}\tau \vdash K$	by subderivation lemma
$\Psi \vdash K\{M/x\}$	by substitution

Case: $\mathsf{TEq}\eta$

$\Psi \vdash \sigma :: \Pi x{:}\tau.K$ by inversion
$\Psi \vdash \lambda x{:}\tau.(\sigma\, x) :: \Pi x{:}\tau.K$ by wf rules
$\Psi \vdash \Pi x{:}\tau.K$ by subderivation lemma

Case: $\mathsf{TEq}\{\}$

Straightforward by i.h.

Case: (3) is identical to (2), appealing to inversion for $\forall\exists$ as needed.
Case: $\mathsf{PEqRefl}$

Immediate + subderivation lemma.

Case: PEqT and PEqS

i.h.

Case: PEqRed

$\Psi;\Gamma;\Delta \vdash P :: z{:}A$, $P \to Q$ and $\Psi;\Gamma;\Delta \vdash Q :: z{:}A$ by inversion
$\Psi \vdash A :: \mathsf{stype}$ by subderivation lemma

Case: $\mathsf{PEq}\forall\mathsf{R}$

Straightforward by i.h.

Case: $\mathsf{PEq}\forall\mathsf{L}$

$\Psi \vdash M_0 = M_1 : \tau$ and $\Psi;\Gamma;\Delta, x{:}A\{M_0/y\} \vdash P_0 = Q_0 :: z{:}C$ by inversion
$\Psi;\Gamma;\Delta, x{:}A\{M_0/y\} \vdash P_0 :: z{:}C$, $\Psi;\Gamma;\Delta, x{:}A\{M_0/y\} \vdash Q_0 :: z{:}C$
and $\Psi \vdash C :: \mathsf{stype}$ by i.h.
$\Psi \vdash M_0 : \tau$, $\Psi \vdash M_1 : \tau$ and $\Psi \vdash \tau :: \mathsf{type}$ by i.h.
$\Psi;\Gamma;\Delta, x{:}\forall y{:}\tau.A \vdash x\langle M_0\rangle.P_0 :: z{:}C$ by $\forall\mathsf{L}$
$\Psi;\Delta, x{:}\forall y{:}\tau.A \vdash$ by subderivation lemma
$\Psi \vdash \forall y{:}\tau.A :: \mathsf{stype}$ by definition
$\Psi, y{:}\tau \vdash A :: \mathsf{stype}$ by inversion for $\forall\exists$
$\Psi \vdash A\{M_0/y\} = A\{M_1/y\} :: \mathsf{stype}$ by functionality
$\Psi \vdash A\{M_1/y\} :: \mathsf{stype}$ by substitution
$\Psi;\Gamma;\Delta, x{:}A\{M_1/y\} \vdash Q_0 :: z{:}C$ by context conversion rule
$\Psi;\Gamma;\Delta, x{:}\forall y{:}\tau.A \vdash x\langle M_1\rangle.Q_0 :: z{:}C$ by $\forall\mathsf{L}$

Case: $\mathsf{PEqConvR}$

$\Psi;\Gamma;\Delta \vdash P = Q :: z{:}A$ and $\Psi \vdash A = B :: \mathsf{stype}$ by inversion
$\Psi;\Gamma;\Delta \vdash P :: z{:}A$, $\Psi;\Gamma;\Delta \vdash Q :: z{:}A$,
$\Psi \vdash A :: \mathsf{stype}$ and $\Psi \vdash B :: \mathsf{stype}$ by i.h.
$\Psi;\Gamma;\Delta \vdash P :: z{:}B$ by $\mathsf{PEqConvR}$
$\Psi;\Gamma;\Delta \vdash Q :: z{:}B$ by $\mathsf{PEqConvR}$

Remaining cases are identical.

Theorem B.10 (Functionality for Equality). *Assume $\Psi \vdash M = N : \tau$:*

1. *If $\Psi, x{:}\tau \vdash M_0 = M_1 : \sigma$ then $\Psi \vdash M_0\{M/x\} = M_1\{N/x\} : \sigma\{M/x\}$*
2. *If $\Psi, x{:}\tau \vdash \sigma_1 = \sigma_2 :: K$ then $\Psi \vdash \sigma_1\{M/x\} = \sigma_2\{N/x\} :: K\{M/x\}$*
3. *If $\Psi, x{:}\tau \vdash A = B :: K$ then $\Psi \vdash A\{M/x\} = B\{N/x\} :: K\{M/x\}$*
4. *If $\Psi, x{:}\tau \vdash K_1 = K_2$ then $\Psi \vdash K_1\{M/x\} = K_2\{N/x\}$*
5. *If $\Psi, x{:}\tau; \Gamma; \Delta \vdash P = Q :: z{:}A$ then $\Psi; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = Q\{N/x\} :: z{:}A\{M/x\}$*

Proof. **(1)**

$\Psi, x{:}\tau \vdash M_0 = M_1 : \sigma$ assumption

$\Psi \vdash M = N : \tau$ assumption

$\Psi \vdash M : \tau$, $\Psi \vdash N : \tau$ and $\Psi \vdash \tau :: \mathsf{type}$ by validity

$\Psi, x{:}\tau \vdash M_0 : \sigma$, $\Psi, x{:}\tau \vdash M_1 : \sigma$ and $\Psi, x{:}\tau \vdash \sigma :: \mathsf{type}$ by validity

$\Psi \vdash M_0\{M/x\} = M_1\{M/x\} : \sigma\{M/x\}$ by substitution

$\Psi \vdash M_1\{M/x\} = M_1\{N/x\} : \sigma\{M/x\}$ by functionality

$\Psi \vdash M_0\{M/x\} = M_1\{N/x\} : \sigma\{M/x\}$ by transitivity

(2)

$\Psi, x{:}:\tau; \Gamma; \Delta \vdash P = Q :: z{:}A$ assumption

$\Psi \vdash M = N : \tau$ assumption

$\Psi \vdash M : \tau$, $\Psi \vdash N : \tau$ and $\Psi \vdash \tau :: \mathsf{type}$ by validity

$\Psi, x{:}\tau; \Gamma; \Delta \vdash P :: z{:}A$, $\Psi, x{:}\tau; \Gamma; \Delta \vdash Q :: z{:}A$
and $\Psi, x{:}\tau \vdash A :: \mathsf{stype}$ by validity

$\Psi; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = Q\{M/x\} :: z{:}A\{M/x\}$ by substitutition

$\Psi; \Gamma\{M/x\}; \Delta\{M/x\} \vdash Q\{M/x\} = Q\{N/x\} :: z{:}A\{M/x\}$ by functionality

$\Psi; \Gamma\{M/x\}; \Delta\{M/x\} \vdash P\{M/x\} = Q\{N/x\} :: z{:}A\{M/x\}$ by transitivity

Remaining cases are identical, appealing to validity, substitution and functionality of typing.

We omit the analogue functionality property for type substitution.

Lemma B.11 (Inversion).

1. *If $\Psi \vdash x{:}\tau$ then $x{:}\sigma \in \Psi$ with $\Psi \vdash \tau = \sigma :: \mathsf{type}$*
2. *If $\Psi \vdash M_1\, M_2 : \sigma$ then $\Psi \vdash M_1 : \Pi x{:}\tau_1.\tau_2$, $\Psi \vdash M_2 : \tau_1$ and $\Psi \vdash \sigma\{M_2/x\} = \tau_2 :: \mathsf{type}$*
3. *If $\Psi \vdash \lambda x{:}\tau.M : \sigma$ then $\Psi \vdash \sigma = \Pi x{:}\tau.\sigma' :: \mathsf{type}$, $\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash M : \sigma'$*
4. *If $\Psi \vdash \Pi x{:}\tau_1.\tau_2 :: K$ then $\Psi \vdash K = \mathsf{type}$, $\Psi \vdash \tau_1 :: \mathsf{type}$ and $\Psi, x{:}\tau_1 \vdash \tau_2 :: \mathsf{type}$*
5. *If $\Psi \vdash \lambda x{:}\tau.\sigma :: K$ then $\Psi \vdash K = \Pi x{:}\tau.K'$, $\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \sigma :: K'$*
6. *If $\Psi \vdash \tau\, M :: K$ then $\Psi \vdash \tau :: \Pi x{:}\tau_0.K_1$, $\Psi \vdash M : \tau_0$ and $\Psi \vdash K = K_1\{M/x\}$*

7. *If* $\Psi \vdash \lambda t{::}K.\tau :: K'$ *then* $\Psi \vdash K' = \Pi t{::}K.K''$, $\Psi \vdash K$ *and* $\Psi, t{::}K \vdash \tau :: K''$
8. *If* $\Psi \vdash \tau\,\sigma :: K$ *then* $\Psi \vdash \tau :: \Pi t{::}K_0.K_1$, $\Psi \vdash \sigma : K_0$ *and* $\Psi \vdash K = K_1\{\sigma/t\}$
9. *If* $\Psi \vdash \Pi x{:}\tau.K$ *then* $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash K$
10. *If* $\Psi \vdash \Pi t{::}K_1.K_2$ *then* $\Psi \vdash K_1$ *and* $\Psi, t{::}K_1 \vdash K_2$
11. *If* $\Psi \vdash \{\Gamma; \Delta \vdash c{:}A\} :: K$ *then* $\Psi \vdash K = \mathsf{type}$, $\Psi \vdash \Gamma :: \mathsf{stype}$, $\Psi \vdash \Delta :: \mathsf{stype}$ *and* $\Psi \vdash A :: \mathsf{stype}$
12. *If* $\Psi; \Gamma; \Delta \vdash z(x{:}\tau).P :: z{:}A$ *then* $\Psi \vdash A = \forall x{:}\tau.A'$ *and* $\Psi \vdash \tau :: \mathsf{stype}$ *and* $\Psi, x{:}\tau; \Gamma; \Delta \vdash P :: z{:}A'$
13. *If* $\Psi; \Gamma; \Delta, x{:}A \vdash x\langle M\rangle_{\forall x:\tau.A'}.P :: z{:}C$ *then* $\Psi \vdash A = \forall y{:}\tau.A' :: \mathsf{stype}$, $\Psi \vdash \tau :: \mathsf{type}$, $\Psi \vdash M : \tau$ *and* $\Psi; \Gamma; \Delta, x{:}A'\{M/y\} \vdash P :: z{:}C$
14. *If* $\Psi; \Gamma; \Delta \vdash z\langle M\rangle_{\exists x:\tau.A'}.P :: z{:}A$ *then* $\Psi \vdash A = \exists x{:}\tau.A' :: \mathsf{stype}$, $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, y{:}\tau; \Gamma; \Delta, x{:}A' \vdash P :: z{:}C$
15. *If* $\Psi \vdash \forall x{:}\tau.A :: K$ *then* $\Psi \vdash K = \mathsf{stype}$, $\Psi \vdash \tau :: \mathsf{type}$, $\Psi, x{:}\tau \vdash A :: \mathsf{stype}$
16. *If* $\Psi \vdash \exists x{:}\tau.A :: K$ *then* $\Psi \vdash K = \mathsf{stype}$, $\Psi \vdash \tau :: \mathsf{type}$, $\Psi, x{:}\tau \vdash A :: \mathsf{stype}$
17. *If* $\Psi \vdash \lambda x{:}\tau.A :: K$ *then* $\Psi \vdash K = \Pi x{:}\tau.K'$, $\Psi \vdash \tau :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash A :: K'$
18. *If* $\Psi \vdash A\,M :: K$ *then* $\Psi \vdash A :: \Pi x{:}\tau_0.K'$, $\Psi \vdash M : \tau_0$ *and* $\Psi \vdash K = K'\{M/x\}$
19. *If* $\Psi \vdash \lambda t{::}K.A :: K'$ *then* $\Psi \vdash K' = \Pi t{::}K.K''$, $\Psi \vdash K$ *and* $\Psi, t{::}K \vdash A :: K''$
20. *If* $\Psi \vdash A\,B :: K$ *then* $\Psi \vdash A :: \Pi t{:}K_0.K_1$, $\Psi \vdash B :: K_0$ *and* $\Psi \vdash K = K_1\{B/t\}$

Proof. By induction on the given derivation. Most cases require validity.

Theorem B.12 (Equality Inversion).

1. *If* $\Psi \vdash \tau = \Pi x{:}\tau_0.\tau_1 :: \mathsf{type}$ *then* $\Psi \vdash \tau = \Pi x{:}\sigma_0.\sigma_1 :: \mathsf{type}$ *with* $\Psi \vdash \sigma_0 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\sigma_0 \vdash \sigma_1 = \tau_1 :: \mathsf{type}$
2. *If* $\Psi \vdash K = \mathsf{type}$ *then* $K = \mathsf{type}$
3. *If* $\Psi \vdash K = \Pi x{:}\tau_0.K'$ *then* $\Psi \vdash K = \Pi x{:}\sigma_0.K''$ *with* $\Psi \vdash \sigma_0 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\sigma_0 \vdash K'' = K'$
4. *If* $\Psi \vdash K = \Pi t{::}K_1.K_2$ *then* $\Psi \vdash K = \Pi t{::}K_1'.K_2'$ *with* $\Psi \vdash K_1' = K_1$ *and* $\Psi, t{::}K_1' \vdash K_2' = K_2$
5. $\Psi \vdash A = \forall x{:}\tau_0.A_0 :: \mathsf{stype}$ *then* $\Psi \vdash A = \forall x{:}\sigma_0.B_0 :: \mathsf{stype}$ *with* $\Psi \vdash \sigma_0 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\sigma_0 \vdash B_0 = A_0 :: \mathsf{stype}$
6. $\Psi \vdash A = \exists x{:}\tau_0.A_0 :: \mathsf{stype}$ *then* $\Psi \vdash A = \exists x{:}\sigma_0.B_0 :: \mathsf{stype}$ *with* $\Psi \vdash \sigma_0 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\sigma_0 \vdash B_0 = A_0 :: \mathsf{stype}$
7. $\Psi \vdash \tau = \lambda x{:}\tau_0.\sigma :: K$ *then* $\Psi \vdash \tau = \lambda x{:}\tau_1.\sigma' :: \Pi x{:}\tau_1.K_0$ *with* $\Psi \vdash \tau_1 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\tau_1 \vdash \sigma' = \sigma :: K_0$, *for some* K_0
8. $\Psi \vdash \tau = \tau_0\,M :: K$ *then* $\Psi \vdash \tau = \tau_1\,N :: K$ *with* $\Psi \vdash \tau_1 = \tau_0 :: \Pi x{:}\sigma.K_0$, $\Psi \vdash N = M : \sigma$ *and* $K = K_0\{N/x\}$
9. $\Psi \vdash \tau = \lambda t{::}K.\sigma :: K'$ *then* $\Psi \vdash \tau = \lambda t{::}K_0.\sigma' :: \Pi t{::}K_0.K''$ *with* $\Psi \vdash K_0 = K$ *and* $\Psi, t{::}K_0 \vdash \sigma' = \sigma :: K''$, *for some* K''
10. $\Psi \vdash \tau = \tau_0\,\sigma_0 :: K$ *then* $\Psi \vdash \tau = \tau_1\,\sigma_1 :: K$ *with* $\Psi \vdash \tau_1 = \tau_0 :: \Pi t{::}K_1.K_0$, $\Psi \vdash \sigma_1 = \sigma_0 :: K_1$ *and* $K = K_0\{\sigma_1/t\}$
11. $\Psi \vdash A = \lambda x{:}\tau_0.A_0 :: K$ *then* $\Psi \vdash A = \lambda x{:}\tau_1.A_0' :: \Pi x{:}\tau_1.K_0$ *with* $\Psi \vdash \tau_1 = \tau_0 :: \mathsf{type}$ *and* $\Psi, x{:}\tau_1 \vdash A_0' = A_0 :: K_0$, *for some* K_0
12. $\Psi \vdash A = A_0\,M :: K$ *then* $\Psi \vdash A = A_1\,N$ *with* $\Psi \vdash A_1 = A_0 :: \Pi x{:}\sigma.K_0$, $\Psi \vdash N = M : \sigma$ *and* $K = K_0\{N/x\}$

13. *If* $\Psi \vdash B = \lambda t{::}K.A :: K'$ *then* $\Psi \vdash B = \lambda t{::}K_0.A' :: \Pi t{::}K_0.K''$ *with* $\Psi \vdash K_0 = K$ *and* $\Psi, t{::}K_0 \vdash A' = A :: K''$*, for some* K''
14. $\Psi \vdash A = A_0\,B_0 :: K$ *then* $\Psi \vdash A = A_1\,B_1 :: K$ *with* $\Psi \vdash A_1 = A_0 :: \Pi t{::}K_1.K_0$*,* $\Psi \vdash B_1 = B_0 :: K_1$ *and* $K = K_0\{B_1/t\}$

Proof. By induction on the given equality derivations.
(1)

Case: TEqT

$\Psi \vdash \tau = \tau' :: \mathsf{type}$ and $\Psi \vdash \tau' = \Pi x{:}\tau_0.\tau_1 :: \mathsf{type}$ assumption
$\tau' = \Pi x{:}\tau_0'.\tau_1'$ with $\Psi \vdash \tau_0' = \tau_0 :: \mathsf{type}$ and $\Psi, x{:}\tau_0' \vdash \tau_1' = \tau_1 :: \mathsf{type}$ by i.h.
$\tau = \Pi x{:}\sigma_0.\sigma_1$ with $\Psi \vdash \sigma_0 = \tau_0' :: \mathsf{type}$ and $\Psi, x{:}\sigma_0 \vdash \sigma_1 = \tau_1' :: \mathsf{type}$ by i.h.
$\Psi \vdash \sigma_0 = \tau_0 :: \mathsf{type}$ by transivitity
$\Psi, x{:}\sigma_0 \vdash \tau_1' = \tau_1 :: \mathsf{type}$ by context conversion
$\Psi, x{:}\sigma_0 \vdash \sigma_1 = \tau_1 :: \mathsf{type}$ by transitivity

Case: $\mathsf{TEq}\beta$

$\Psi, y{:}\tau \vdash \sigma :: \mathsf{type}$ and $\Psi \vdash M : \tau,\ K\{M/y\} = \mathsf{type}$ and
$\Pi x{:}\tau_0.\tau_1 = \sigma\{M/y\}$ by inversion
$\Psi \vdash (\lambda y{:}\tau.\sigma)\,M = \Pi x{:}\tau_0.\tau_1 :: \mathsf{type}$ assumption
$\sigma = \Pi x{:}\tau.\sigma$ by definition of substitution
$\Psi, y{:}\tau \vdash \Pi x{:}\tau.\sigma :: \mathsf{type}$ by def.
$\Psi \vdash (\lambda y{:}\tau.\sigma)\,M = \Pi x{:}\tau.\sigma\{M/y\} :: \mathsf{type}$ by rule
$\Psi \vdash \tau :: \mathsf{type}$ by validity
$\Psi \vdash \tau = \tau :: \mathsf{type}$ by reflexivity
$\Psi \vdash \sigma\{M/y\} :: \mathsf{type}$ by substitution
$\Psi \vdash \sigma\{M/y\} = \sigma\{M/y\} :: \mathsf{type}$ by reflexivity

The other cases follow similar patterns.

Lemma B.13 (Injectivity of Products).

1. *If* $\Psi \vdash \Pi x{:}\tau.\sigma = \Pi x{:}\tau'.\sigma' :: \mathsf{type}$ *then* $\Psi \vdash \tau = \tau' :: \mathsf{type}$ *and* $\Psi, x{:}\tau \vdash \sigma = \sigma' :: \mathsf{type}$
2. *If* $\Psi \vdash \Pi x{:}\tau_1.K_1 = \Pi x{:}\tau_2.K_2$ *then* $\Psi \vdash \tau_1 = \tau_2 :: \mathsf{type}$ *and* $\Psi, x{:}\tau_1 \vdash K_1 = K_2$
3. *If* $\Psi \vdash \forall x{:}\tau_1.A_1 = \forall x{:}\tau_2.A_2 :: \mathsf{stype}$ *then* $\Psi \vdash \tau_1 = \tau_2 :: \mathsf{type}$ *and* $\Psi, x{:}\tau_1 \vdash A_1 = A_2 :: \mathsf{stype}$

Proof. By equality inversion.

Theorem 2.4 (Unicity of Types and Kinds).

1. *If* $\Psi \vdash M : \tau$ *and* $\Psi \vdash M : \tau'$ *then* $\Psi \vdash \tau = \tau' :: \mathsf{type}$
2. *If* $\Psi \vdash \tau :: K$ *and* $\Psi \vdash \tau :: K'$ *then* $\Psi \vdash K = K'$
3. *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *and* $\Psi; \Gamma; \Delta \vdash P :: z{:}A'$ *then* $\Psi \vdash A = A' :: \mathsf{stype}$
4. *If* $\Psi \vdash A :: K$ *and* $\Psi \vdash A :: K'$ *then* $\Psi \vdash K = K'$

Proof. By induction on the structure of the given term/type/process.

Case: M is $\lambda x{:}\tau_0.M'$

$\Psi, x{:}\tau_0 \vdash M' : \sigma$, $\Psi, x{:}\tau_0 \vdash M' : \sigma'$ with $\Psi \vdash \tau = \Pi x{:}\tau_0.\sigma :: \mathsf{type}$,
$\Psi \vdash \tau_0 :: \mathsf{type}$ and $\Psi \vdash \tau' = \Pi x{:}\tau_0.\sigma' :: \mathsf{type}$ by inversion
$\Psi, x{:}\tau_0 \vdash \sigma = \sigma' :: \mathsf{type}$ by i.h.
$\Psi \vdash \Pi x{:}\tau_0.\sigma = \Pi x{:}\tau_0.\sigma' :: \mathsf{type}$ by $\mathsf{TEq}\Pi$ rule

Case: M is $M'\,N'$

$\Psi \vdash M'N' : \tau$ and $\Psi \vdash M'N' : \tau'$ assumption
$\Psi \vdash M' : \Pi x{:}\tau_0.\sigma_0$ and $\Psi \vdash M' : \Pi x{:}\tau_0'.\sigma_0'$ with $\Psi \vdash \tau = \sigma_0\{N'/x\} :: \mathsf{type}$,
$\Psi \vdash N' : \tau_0$, $\Psi \vdash N' : \tau_0'$ and $\Psi \vdash \tau' = \sigma_0'\{N'/x\} :: \mathsf{type}$ by inversion
$\Psi \vdash \Pi x{:}\tau_0.\sigma_0 = \Pi x{:}\tau_0'.\sigma_0' :: \mathsf{type}$ by i.h.
$\Psi \vdash \tau_0 = \tau_0' :: \mathsf{type}$ by i.h.
$\Psi, x{:}\tau_0 \vdash \sigma_0 = \sigma_0' :: \mathsf{type}$ by injectivity
$\Psi \vdash \sigma_0\{N'/x\} = \sigma_0'\{N'/x\} :: \mathsf{type}$ by functionality

Case: M is $\{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$

$\Psi \vdash M : \{\Gamma; \Delta \vdash c{:}A\}$ and $\Psi \vdash M : \{\Gamma; \Delta \vdash c{:}A'\}$ assumption
$\Psi; \Gamma; \Delta \vdash P :: c{:}A$ and $\Psi; \Gamma; \Delta \vdash P :: c{:}A'$ by inversion
$\Psi \vdash A = A' :: \mathsf{stype}$ by i.h.
Conclude by reflexivity and $\mathsf{TEq}\{\}$

Case: M is x
Direct by inversion lemma.
(2)
Case: τ is $\Pi x{:}\tau_0.\sigma_0$

$\Psi \vdash \Pi x{:}\tau_0.\sigma_0 :: K$ and $\Psi \vdash \Pi x{:}\tau_0.\sigma_0 :: K'$ assumption
$\Psi, x{:}\tau_0 \vdash \sigma_0 :: \mathsf{type}$ and $\Psi, x{:}\tau_0 \vdash \sigma_0 :: \mathsf{type}$ and $K = K' = \mathsf{type}$
by inversion lemma

Case: τ is $\lambda x{:}\tau_0.\sigma_0$

$\Psi \vdash \lambda x{:}\tau_0.\sigma_0 :: K$ and $\Psi \vdash \lambda x{:}\tau_0.\sigma_0 :: K'$ assumption
$\Psi, x{:}\tau_0 \vdash \sigma_0 :: K_0$, $\Psi, x{:}\tau_0 \vdash \sigma_0 :: K_0'$, $\Psi \vdash \tau_0 :: \mathsf{type}$
with $K = \Pi x{:}\tau_0.K_0$ and $K' = \Pi x{:}\tau_0.K_0'$ by inversion lemma
$\Psi, x{:}\tau_0 \vdash K_0 = K_0'$ by i.h.
$\Psi \vdash \Pi x{:}\tau_0.K_0 = \Pi x{:}\tau_0.K_0'$ by rule

Case: τ is $\tau_0\,M$

$\Psi \vdash \tau_0\,M :: K$ and $\Psi \vdash \tau_0\,M :: K'$ assumption
$\Psi \vdash \tau_0 :: \Pi x{:}\sigma.K_0$ and $\Psi \vdash \tau_0 :: \Pi x{:}\sigma'.K_0'$, $\Psi \vdash M : \sigma$ and $\Psi \vdash M : \sigma'$
with $K = K_0\{M/x\}$ and $K' = K_0'\{M/x\}$ by inversion lemma
$\Psi \vdash \Pi x{:}\sigma.K_0 = \Pi x{:}\sigma'.K_0'$ by i.h.
$\Psi \vdash \sigma = \sigma' :: \mathsf{type}$ by i.h.
$\Psi, x{:}\sigma \vdash K_0 = K_0'$ by injectivity
$\Psi \vdash K_0\{M/x\} = K_0'\{M/x\}$ by substitution

Case: τ is $\{\Gamma; \Delta \vdash c{:}A\}$

Straightforward by i.h.

(3)

Case: P is $z(x).P_0$

$\Psi; \Gamma; \Delta \vdash z(x{:}\tau_0).P_0 :: z{:}A$ and $\Psi; \Gamma; \Delta \vdash z(x{:}\tau_0).P_0 :: z{:}A'$ assumption
$A = \forall x{:}\tau_0.A_0,\ A' = \forall x{:}\tau_0.A_0',\ \Psi, x{:}\tau_0; \Gamma; \Delta \vdash P_0 :: z{:}A_0,$
$\Psi, x{:}\tau_0; \Gamma; \Delta \vdash P_0 :: z{:}A_0'$, and $\Psi \vdash \tau_0 :: \mathsf{type}$ by inversion lemma
$\Psi, x{:}\tau_0 \vdash A_0 = A_0'$ by i.h.
$\Psi \vdash \forall x{:}\tau_0 A_0 = \forall x{:}\tau_0 A_0'$ by rule

Case: P is $x\langle M\rangle_{\forall x:\tau_0.A_0}.P_0$

$\Psi; \Gamma; \Delta, x{:}A \vdash x\langle M\rangle_{\forall x:\tau_0.A_0}.P_0 :: z{:}C$ and
$\Psi; \Gamma; \Delta, x{:}A \vdash x\langle M\rangle.P_0 :: z{:}C$ assumption
$A = \forall x{:}\tau_0.A_0,\ \Psi \vdash M : \tau_0,\ \Psi; \Gamma; \Delta, x{:}A_0\{M/x\} \vdash P_0 :: z{:}C$
and $\Psi; \Gamma; \Delta, x{:}A_0\{M/x\} \vdash P_0 :: z{:}C'$ by inversion lemma
$\Psi \vdash C = C' :: \mathsf{stype}$ by i.h.

Case: P is $z\langle M\rangle_{\exists x:\tau_0.A_0}.P_0$

$\Psi; \Gamma; \Delta \vdash z\langle M\rangle.P_0 :: z{:}A$ and $\Psi; \Gamma; \Delta \vdash z\langle M\rangle.P_0 :: z{:}A'$ assumption
$A = \exists x{:}\tau_0.A_0,\ A' = \exists x{:}\tau_0.A_0,\ \Psi \vdash M : \tau_0,$
$\Psi; \Gamma; \Delta \vdash P_0 :: z{:}A_0\{M/x\}$ and $\Psi; \Gamma; \Delta \vdash P_0 :: z{:}A_0\{M/x\}$
by inversion lemma

Remaining cases follow similarly.

Theorem B.14. *If* $\Psi \vdash M : \tau$ *and* $M \to M'$ *then* $\Psi \vdash M = M' : \tau$

Proof. By induction on $\to$ relation.

Case:

$$\frac{M \to M'}{M\, N \to M'\, N}$$

$\Psi \vdash M : \Pi x{:}\tau_0.\sigma_0,\ \Psi \vdash \tau_0 :: \mathsf{type},\ \Psi \vdash N : \tau_0$ and $\Psi \vdash M\, N : \sigma_0\{N/x\}$
by inversion lemma
$\Psi \vdash M = M' : \Pi x{:}\tau_0.\sigma_0$ by i.h.
$\Psi \vdash \Pi x{:}\tau_0.\sigma_0 :: \mathsf{type}$ by validity
$\Psi \vdash N = N : \tau_0$ by reflexivity
$\Psi \vdash M\, N = M'\, N : \sigma_0\{N/x\}$ by $\mathsf{TMEq}\Pi$

Case:

$$\frac{N \to N'}{M\, N \to M\, N'}$$

$\Psi \vdash M : \Pi x{:}\tau_0.\sigma_0$, $\Psi \vdash \tau_0 :: \mathsf{type}$, $\Psi \vdash N : \tau_0$ and $\Psi \vdash M\,N : \sigma_0\{N/x\}$
by inversion lemma

$\Psi \vdash N = N' : \tau_0$ by i.h.

$\Psi \vdash M = M : \Pi x{:}\tau_0.\sigma_0$ by reflexivity

$\Psi \vdash M\,N = M\,N' : \sigma_0\{N/x\}$ by $\mathsf{TMEq}\Pi$

$$\frac{}{(\lambda x{:}\tau_0.M_0)\,N_0 \to M_0\{N_0/x\}}$$

$\Psi \vdash \lambda x{:}\tau_0.M_0 : \Pi x{:}\tau_0.\sigma_0$, $\Psi \vdash \tau_0 :: \mathsf{type}$, $\Psi \vdash N_0 : \tau_0$,
$\Psi \vdash (\lambda x{:}\tau_0.M_0)\,N_0 : \sigma_0\{N_0/x\}$ by inversion lemma

$\Psi, x{:}\tau_0 \vdash M_0 : \sigma_0$ by inversion lemma

$\Psi \vdash (\lambda x{:}\tau_0.M_0)\,N_0 = M_0\{N_0/x\} : \sigma_0\{N/x\}$ by $\mathsf{TMEq}\beta$

Theorem 2.5 (Subject Reduction – Terms). *If $\Psi \vdash M : \tau$ and $M \to M'$ then $\Psi \vdash M' : \tau$*

Proof. Immediate from Theorem B.14 and validity for equality.

Theorem 2.6 (Subject Reduction – Processes). *If $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ and $P \to P'$ then $\exists Q$ such that $P' \equiv Q$ and $\Psi; \Gamma; \Delta \vdash Q :: z{:}A$*

Proof. The proof follows by Theorem B.14 and a series of lemmas that relate typed processes and their reducts under a cut (which now crucially rely on the inversion lemmas and validity). See [29,28,5].

Theorem 2.7 (Progress – Terms). *If $\Psi \vdash M : \tau$ then either M is a value or $M \to M'$*

Proof. By induction on typing, using the standard canonical forms-based reasoning and noting that monadic terms are values.

C Appendix – Embedding

Lemma C.1 (Compositionality).

1. $\Psi \vdash \llbracket K\{M/x\}\rrbracket$ *iff* $\Psi \vdash \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
2. $\Psi \vdash \llbracket K_1\{\tau/t\}\rrbracket$ *iff* $\Psi \vdash \llbracket K_1\rrbracket\{\llbracket\tau\rrbracket/t\}$
3. $\Psi \vdash \llbracket K_1\{A/x\}\rrbracket$ *iff* $\Psi \vdash \llbracket K_1\rrbracket\{\llbracket A\rrbracket/x\}$
4. $\Psi \vdash \llbracket\tau\{M/x\}\rrbracket :: \llbracket K\{M/x\}\rrbracket$ *iff* $\Psi \vdash \llbracket\tau\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
5. $\Psi \vdash \llbracket A\{M/x\}\rrbracket :: \llbracket K\{M/x\}\rrbracket$ *iff* $\Psi \vdash \llbracket A\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
6. $\Psi; \Gamma; \Delta \vdash \llbracket M\{N/x\}\rrbracket_z = \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_y\}/x\} :: z{:}\llbracket A\{N/x\}\rrbracket$
7. $\Psi; \Gamma; \Delta \vdash \llbracket P\{M/x\}\rrbracket :: z{:}\llbracket A\{M/x\}\rrbracket$ *iff* $\Psi; \Gamma; \Delta \vdash \llbracket P\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: z{:}\llbracket A\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$

Proof. By mutual induction on the structure of the given kind/type/session type/etc.

Case: $K = \mathsf{type}$ or $K = \mathsf{stype}$

Trivial.

(1)

Case: $K = \Pi y{:}\tau.K'$

Subcase: $\Rightarrow$
$\llbracket \Pi y{:}\tau.K'\{M/x\}\rrbracket = \llbracket \Pi y{:}\tau\{M/x\}.K'\{M/x\}\rrbracket = \Pi y{:}\{\llbracket \tau\{M/x\}\rrbracket\}.\llbracket K'\{M/x\}\rrbracket$ by definition
$\Pi y{:}\{\llbracket \tau\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket K'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by i.h.(3) and i.h.(1)
$= (\Pi y{:}\{\llbracket \tau\rrbracket\}.\llbracket K'\rrbracket)\{\{\llbracket M\rrbracket_c\}/x\}$ by definition, satisfying $\Rightarrow$
Subcase: $\Leftarrow$
$\llbracket \Pi y{:}\tau.K'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = (\Pi y{:}\{\llbracket \tau\rrbracket\}.\llbracket K'\rrbracket)\{\{\llbracket M\rrbracket_c\}/x\} =$
$\Pi y{:}\{\llbracket \tau\rrbracket\}\{\{\llbracket M\rrbracket_c\}/x\}.\llbracket K'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by definition
$\Pi y{:}\{\llbracket \tau\{M/x\}\rrbracket\}.\llbracket K'\{M/x\}\rrbracket$ by i.h.(3) and i.h.(1)
$= \llbracket \Pi y{:}\tau.K'\{M/x\}\rrbracket$ by definition, satisfying $\Leftarrow$

Case: $K = \Pi t{:}K_1.K_2$

Same argument as above, appealing to i.h.(1)

(2)
As above, appealing to i.h.(2)
(3)
As above, appealing to i.h.(3)
(4)
Case: $\tau = \Pi y{:}\tau'.\sigma$

Subcase: $\Rightarrow$
$\llbracket \Pi y{:}\tau'.\sigma\{M/x\}\rrbracket = \llbracket \Pi y{:}\tau'\{M/x\}.\sigma\{M/x\}\rrbracket = \forall y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket$ by definition
$\forall y{:}\{\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by i.h.(3)
$= (\forall y{:}\{\llbracket \tau'\rrbracket\}.\llbracket \sigma\rrbracket)\{\{\llbracket M\rrbracket_c\}/x\}$ by definition, satisfying $\Rightarrow$
Subcase: $\Leftarrow$
$\llbracket \Pi y{:}\tau'.\sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = (\forall y{:}\{\llbracket \tau'\rrbracket\}.\llbracket \sigma\rrbracket)\{\{\llbracket M\rrbracket_c\}/x\} = \forall y{:}\{\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
by definition
$\forall y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket$ by i.h.(3)
$= \llbracket \Pi y{:}(\tau'\{M/x\}).(\sigma\{M/x\})\rrbracket = \llbracket \Pi y{:}\tau'.\sigma\{M/x\}\rrbracket$ by definition, satisfying $\Leftarrow$

Case: $\tau = \lambda y{:}\tau'.\sigma$

As above.

Case: $\tau = \tau'\, M'$

$\llbracket \tau'\, M'\{M/x\}\rrbracket = \llbracket \tau'\{M/x\}\, M'\{M/x\}\rrbracket = \llbracket \tau'\{M/x\}\rrbracket\,\{\llbracket M'\{M/x\}\rrbracket_c\}$ by definition
$\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_d\}/x\}\,\{\llbracket M'\rrbracket_c\{\{\llbracket M\rrbracket_d\}/x\}\}$ by i.h.
$\llbracket \tau'\, M'\rrbracket\{\{\llbracket M\rrbracket_d\}/x\} = (\llbracket \tau'\rrbracket\,\{\llbracket M'\rrbracket_c\})\{\{\llbracket M\rrbracket_d\}/x\} = \llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_d\}/x\}\,\{\llbracket M'\rrbracket_c\{\{\llbracket M\rrbracket_d\}/x\}\}$
by definition

Case: $\tau = \lambda y :: K.\tau'$

$\llbracket \lambda y :: K.\tau'\{M/x\} \rrbracket = \lambda y :: L\{M/x\}.\tau'\{M/x\} \rrbracket = \lambda y :: \llbracket K\{M/x\} \rrbracket.\llbracket \tau'\{M/x\} \rrbracket$ by definition
$\llbracket \lambda y :: K.\tau' \rrbracket\{\{M\}_c/x\} = \lambda y :: \llbracket K \rrbracket\{\{M\}_c/x\}.\llbracket \tau' \rrbracket\{\{M\}_c/x\}$ by definition
$= \lambda y :: \llbracket K\{M/x\} \rrbracket.\llbracket \tau'\{M/x\} \rrbracket$ by i.h.

Case: $\tau = \tau'\,\sigma$

Straightforward by i.h. as above.

(5)

Case: $A = \mathbf{1}$

Trivial.

Case: $A = A_1 \multimap A_2$

$\llbracket A_1 \multimap A_2\{M/x\} \rrbracket = \llbracket A_1\{M/x\} \rrbracket \multimap \llbracket A_2\{M/x\} \rrbracket$ by definition
$\llbracket A_1 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} \multimap \llbracket A_2 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}$ by i.h.
$\llbracket A_1 \multimap A_2 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} = \llbracket A_1 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} \multimap \llbracket A_2 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}$ by definition

Case: $A = A_1 \otimes A_2$

Identical to $\multimap$ case.

Case: $A = \&\{\overline{l_i{:}A_i}\}$

See above.

Case: $A = \oplus\{\overline{l_i{:}A_i}\}$

See above.

Case: $A = \forall x{:}\tau.A_0$

$\llbracket \forall x{:}\tau.A_0\{M/x\} \rrbracket = \forall x{:}\{\llbracket \tau\{M/x\} \rrbracket\}.\llbracket A_0\{M/x\} \rrbracket$ by definition
$\forall x{:}\{\llbracket \tau \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}\}.\llbracket A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}$ by i.h.
$\llbracket \forall x{:}\tau.A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} = \forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} =$
$\forall x{:}\{\llbracket \tau \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}\}.\llbracket A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\}$ by definition

Case: $A = \exists x{:}\tau.A_0$

As above.

Case: $A = \lambda x{:}\tau.A_0$

As above.

Case: $A = A_0\,M'$

$\llbracket A_0\,M'\{M/x\} \rrbracket = \llbracket A_0\{M/x\} \rrbracket\,\{\llbracket M'\{M/x\} \rrbracket_c\}$ by definition
$(\llbracket A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\})\,\{\llbracket M' \rrbracket_d\{\{\llbracket M \rrbracket_c\}/x\}\}$ by i.h.
$\llbracket A_0\,M' \rrbracket\{\{\llbracket M \rrbracket_c\}/x\} = (\llbracket A_0 \rrbracket\,\{\llbracket M' \rrbracket_d\})\{\{\llbracket M \rrbracket_c\}/x\}$
$= (\llbracket A_0 \rrbracket\{\{\llbracket M \rrbracket_c\}/x\})\,\{\llbracket M' \rrbracket_d\{\{\llbracket M \rrbracket_c\}/x\}\}$ by definition

Case: $A = A_0\,A_1$

Straightforward by i.h.

Case: $A = \lambda t :: K.A_0$

Straightforward by i.h.

(6) $\Psi; \Gamma; \Delta \vdash \llbracket M\{N/x\} \rrbracket_z = \llbracket M \rrbracket_z\{\{\llbracket N \rrbracket_y\}/x\} :: z{:}\llbracket A\{N/x\} \rrbracket$

Case: $M = \lambda y{:}\tau.M_0$

$\llbracket \lambda y{:}\tau.M_0\{N/x\}\rrbracket_z :: z{:}\llbracket \Pi y{:}\tau.\sigma\{N/x\}\rrbracket$
$\llbracket \lambda y{:}\tau.M_0\{N/x\}\rrbracket_z = \llbracket \lambda y{:}\tau\{N/x\}.M_0\{N/x\}\rrbracket_z = z(y).\llbracket M_0\{N/x\}\rrbracket_z :: z{:}\forall y{:}\{\llbracket \tau\{N/x\}\rrbracket\}.\llbracket \sigma\{N/x\}\rrbracket$
by definition
$z(y).\llbracket M_0\{N/x\}\rrbracket_z = z(y).\llbracket M_0\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} :: z{:}\forall y{:}\{\llbracket \tau\rrbracket\{\{\llbracket N\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket N\rrbracket_c\}/x\}$ by i.h.
$\llbracket \lambda y{:}\tau.M_0\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = z(y).\llbracket M_0\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\}$ by definition

Case: $M = M_1\, M_2$

$\llbracket M_1\, M_2\{N/x\}\rrbracket_z :: z{:}\llbracket (\sigma\{M_2/y\})\{N/x\}\rrbracket$
$= (\boldsymbol{\nu} y)(\llbracket M_1\{N/x\}\rrbracket_y \mid y\langle\{\llbracket M_2\{N/x\}\rrbracket_y\}\rangle.[y \leftrightarrow z])$ by definition
$= (\boldsymbol{\nu} y)(\llbracket M_1\rrbracket_y\{\{\llbracket N\rrbracket_c\}/x\} \mid y\langle\{\llbracket M_2\rrbracket_y\{\{\llbracket N\rrbracket_c\}/x\}\}\rangle.[y \leftrightarrow z])$ by i.h.
$\llbracket M_1\, M_2\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = (\boldsymbol{\nu} y)(\llbracket M_1\rrbracket_y \mid y\langle\{\llbracket M_2\rrbracket_y\}\rangle.[y \leftrightarrow z])\{\{\llbracket N\rrbracket_c\}/x\}$
$= (\boldsymbol{\nu} y)(\llbracket M_1\rrbracket_y\{\{\llbracket N\rrbracket_c\}/x\} \mid y\langle\{\llbracket M_2\rrbracket_y\{\{\llbracket N\rrbracket_c\}/x\}\}\rangle.[y \leftrightarrow z])$ by definition

Case: $M = \{z \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$

$\llbracket M\{N/x\}\rrbracket_z = z(u_0).\ldots.z(u_j).z(d_0).\ldots.z(d_n).\llbracket P\{N/x\}\rrbracket$ by definition
$z(u_0).\ldots.z(u_j).z(d_0).\ldots.z(d_n).\llbracket P\rrbracket\{\{\llbracket N\rrbracket_c\}/x\}$ by i.h.
$\llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = z(u_0).\ldots.z(u_j).z(d_0).\ldots.z(d_n).\llbracket P\rrbracket\{\{\llbracket N\rrbracket_c\}/x\}$ by definition

Case: $M = y$ with $y \neq x$

$\llbracket y\{N/x\}\rrbracket_z = w \leftarrow y; [w \leftrightarrow z]$ by definition
$\llbracket y\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = w \leftarrow y; [w \leftrightarrow z]$ by definition

Case: $M = y$ with $y = x$

$\llbracket x\{N/x\}\rrbracket_z = \llbracket N\rrbracket_z$ by definition
$\llbracket x\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = w \leftarrow \{\llbracket N\rrbracket_c\}; [w \leftrightarrow z]$ by definition
$w \leftarrow \{\llbracket N\rrbracket_c\}; [w \leftrightarrow z] \to^+ \llbracket N\rrbracket_z$ by reduction semantics
$w \leftarrow \{\llbracket N\rrbracket_c\}; [w \leftrightarrow z] = \llbracket N\rrbracket_z$ by PEqRed

(7) $\Psi; \Gamma; \Delta \vdash \llbracket P\{M/x\}\rrbracket :: z{:}\llbracket A\{M/x\}\rrbracket$ iff $\Psi; \Gamma; \Delta \vdash \llbracket P\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: z{:}\llbracket A\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
Case: $P = (\boldsymbol{\nu} y)(P_1 \mid P_2)$

$\llbracket (\boldsymbol{\nu} y)(P_1 \mid P_2)\{M/x\}\rrbracket = (\boldsymbol{\nu} y)(\llbracket P_1\{M/x\}\rrbracket \mid \llbracket P_2\{M/x\}\rrbracket)$ by definition
$(\boldsymbol{\nu} y)(\llbracket P_1\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} \mid \llbracket P_2\rrbracket\{\{\llbracket M\rrbracket_c\}/x\})$ by i.h.
$\llbracket (\boldsymbol{\nu} y)(P_1 \mid P_2)\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = (\boldsymbol{\nu} y)(\llbracket P_1\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} \mid \llbracket P_2\rrbracket\{\{\llbracket M\rrbracket_c\}/x\})$ by definition

Case: $P = z\langle M_0\rangle.P_0$ by $\exists\mathsf{R}$

$\llbracket z\langle M_0\rangle.P_0\{M/x\}\rrbracket = z\langle\{\llbracket M_0\{M/x\}\rrbracket_d\}\rangle.\llbracket P_0\{M/x\}\rrbracket$ by definition
$z\langle\{\llbracket M_0\rrbracket_d\{\{\llbracket M\rrbracket_c\}/x\}\}\rangle.\llbracket P_0\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by i.h.
$\llbracket z\langle M_0\rangle.P_0\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = z\langle\{\llbracket M_0\rrbracket_d\{\{\llbracket M\rrbracket_c\}/x\}\}\rangle.\llbracket P_0\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by definition

Case: $P = x \leftarrow M_0 \leftarrow \overline{u_j}; \overline{y_i}; P_0$

$$\begin{array}{ll}
\llbracket P\{M/x\}\rrbracket = (\boldsymbol{\nu} c)(\llbracket M_0\{M/x\}\rrbracket_c \mid \overline{c}\langle v_1\rangle.(\overline{u_1}\langle a_1\rangle.[a_1 \leftrightarrow v_1] \mid \cdots \mid & \\
\overline{c}\langle d_1\rangle.([y_1 \leftrightarrow d_1] \mid \cdots \mid \overline{c}\langle d_n\rangle.([y_n \leftrightarrow d_n] \mid \llbracket P_0\{M/x\}\rrbracket) \ldots) & \text{by definition} \\
(\boldsymbol{\nu} c)(\llbracket M_0\rrbracket_c\{\{\llbracket M\rrbracket_c\}/x\} \mid \overline{c}\langle v_1\rangle.(\overline{u_1}\langle a_1\rangle.[a_1 \leftrightarrow v_1] \mid \cdots \mid & \\
\overline{c}\langle d_1\rangle.([y_1 \leftrightarrow d_1] \mid \cdots \mid \overline{c}\langle d_n\rangle.([y_n \leftrightarrow d_n] \mid \llbracket P_0\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}) \ldots) & \text{by i.h.} \\
= \llbracket P\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} & \text{by definition}
\end{array}$$

Remaining process cases are straightforward by i.h.

Lemma C.2 (Compositionality – Reflection in Equality).

1. $\Psi \vdash \llbracket K\{M/x\}\rrbracket = \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$
2. $\Psi \vdash \llbracket K_1\{\tau/t\}\rrbracket = \llbracket K_1\rrbracket\{\llbracket \tau\rrbracket/t\}$
3. $\Psi \vdash \llbracket K_1\{A/x\}\rrbracket = \llbracket K_1\rrbracket\{\llbracket A\rrbracket/x\}$
4. $\Psi \vdash \llbracket \tau\{M/x\}\rrbracket = \llbracket \tau\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\{M/x\}\rrbracket$
5. $\Psi \vdash \llbracket A\{M/x\}\rrbracket = \llbracket A\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\{M/x\}\rrbracket$
6. $\Psi; \Gamma; \Delta \vdash \llbracket M\{N/x\}\rrbracket_z = \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_y\}/x\} :: z{:}\llbracket A\{N/x\}\rrbracket$
7. $\Psi; \Gamma; \Delta \vdash \llbracket P\{M/x\}\rrbracket = \llbracket P\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: z{:}\llbracket A\{M/x\}\rrbracket$

Proof. **(1-3)** is identical to corresponding statements in Lemma C.1.
(4) $\Psi \vdash \llbracket \tau\{M/x\}\rrbracket = \llbracket \tau\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\{M/x\}\rrbracket$

Case: $\tau = \Pi y{:}\tau'.\sigma$

$$\begin{array}{l}
\llbracket \Pi y{:}\tau'.\sigma\{M/x\}\rrbracket = \llbracket \Pi y{:}\tau'\{M/x\}.\sigma\{M/x\}\rrbracket = \forall y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket \text{ by definition} \\
\llbracket \Pi y{:}\tau'.\sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = (\forall y{:}\{\llbracket \tau'\rrbracket\}.\llbracket \sigma\rrbracket)\{\{\llbracket M\rrbracket_c\}/x\} = \forall y{:}\{\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} \\
\qquad \text{by definition} \\
\Psi \vdash \{\llbracket \tau'\{M/x\}\rrbracket\} = \{\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\} :: \mathsf{type} \qquad \text{by i.h. and } \mathsf{TEq}\{\} \\
\Psi, y{:}\{\llbracket \tau'\{M/x\}\rrbracket\} \vdash \llbracket \sigma\{M/x\}\rrbracket = \llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \mathsf{stype} \qquad \text{by i.h.} \\
\Psi \vdash \forall y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket = \forall y{:}\{\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \mathsf{stype} \text{ by } \mathsf{STEq}\forall
\end{array}$$

Case: $\tau = \lambda y{:}\tau'.\sigma$

$$\begin{array}{l}
\llbracket \lambda y{:}\tau'.\sigma\{M/x\}\rrbracket = \lambda y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket \qquad \text{by definition} \\
\llbracket \lambda y{:}\tau'.\sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = \lambda y{:}\{\llbracket \tau'\rrbracket\}\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} \text{ by definition} \\
\Psi \vdash \{\llbracket \tau'\{M/x\}\rrbracket\} = \{\llbracket \tau'\rrbracket\}\{\{\llbracket M\rrbracket_c\}/x\}\} :: \mathsf{type} \qquad \text{by i.h. and } \mathsf{TEq}\{\} \\
\Psi, y{:}\{\llbracket \tau'\{M/x\}\rrbracket\} \vdash \llbracket \sigma\{M/x\}\rrbracket = \llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\{M/x\}\rrbracket \qquad \text{by i.h.} \\
\Psi \vdash \lambda y{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket \sigma\{M/x\}\rrbracket = \lambda y{:}\{\llbracket \tau'\rrbracket\}\{\{\llbracket M\rrbracket_c\}/x\}\}.\llbracket \sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \Pi x{:}\{\llbracket \tau'\{M/x\}\rrbracket\}.\llbracket K\{M/x\}\rrbracket \\
\qquad \text{by } \mathsf{STEq}\lambda
\end{array}$$

Case: $\tau = \tau'\, M'$

$$\begin{array}{l}
\llbracket \tau'\, M'\{M/x\}\rrbracket = \llbracket \tau'\{M/x\}\rrbracket\, \{\llbracket M'\{M/x\}\rrbracket_d\} \qquad \text{by definition} \\
\llbracket \tau'\, M'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} = (\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\})\, \{\llbracket M'\rrbracket_d\{\{\llbracket M\rrbracket_c\}/x\}\} \text{ by definition} \\
\Psi \vdash \llbracket \tau'\{M/x\}\rrbracket = \llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \Pi y{:}\{\llbracket \tau''\{M/x\}\rrbracket\}.\llbracket K\{M/x\}\rrbracket \qquad \text{by i.h} \\
\Psi \vdash \{\llbracket M'\{M/x\}\rrbracket_d\} = \{\llbracket M'\rrbracket_d\{\{\llbracket M\rrbracket_c\}/x\}\} : \{\llbracket \tau''\{M/x\}\rrbracket\} \text{ by i.h. and } \mathsf{TEq}\{\} \\
\Psi \vdash \llbracket \tau'\{M/x\}\rrbracket\, \{\llbracket M'\{M/x\}\rrbracket_d\} = \\
\qquad (\llbracket \tau'\rrbracket\{\{\llbracket M\rrbracket_c\}/x\})\, \{\llbracket M'\rrbracket_d\{\{\llbracket M\rrbracket_c\}/x\}\} :: \llbracket K\{M/x\}\rrbracket\{\{\llbracket M'\{M/x\}\rrbracket_d\}/y\} \text{ by } \mathsf{STEqApp}
\end{array}$$

Case: $\tau = \lambda t :: K'.\tau'$

$\llbracket \lambda t :: K'.\tau'\{M/x\} \rrbracket = \lambda t :: \llbracket K'\{M/x\} \rrbracket . \llbracket \tau'\{M/x\} \rrbracket$ by definition

$\llbracket \lambda t :: K'.\tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} = \lambda t :: \llbracket K' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} . \llbracket \tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\}$ by definition

$\Psi \vdash \llbracket K'\{M/x\} \rrbracket = \llbracket K' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\}$ by i.h.

$\Psi, t :: \llbracket K'\{M/x\} \rrbracket \vdash \llbracket \tau'\{M/x\} \rrbracket = \llbracket \tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} :: \llbracket K''\{M/x\} \rrbracket$ by i.h.

$\Psi \vdash \lambda t :: \llbracket K'\{M/x\} \rrbracket . \llbracket \tau'\{M/x\} \rrbracket =$
$\quad \lambda t :: \llbracket K' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} . \llbracket \tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} :: \Pi t :: \llbracket K'\{M/x\} \rrbracket . \llbracket K''\{M/x\} \rrbracket$ by $\mathsf{STEqT}\lambda$

Case: $\tau = \tau' \, \sigma$

$\llbracket \tau' \, \sigma\{M/x\} \rrbracket = \llbracket \tau'\{M/x\} \rrbracket \, \llbracket \sigma\{M/x\} \rrbracket$ by definition

$\llbracket \tau' \, \sigma \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} = \llbracket \tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} \, \llbracket \sigma \rrbracket \{\{\llbracket M \rrbracket_c\}/x\}$ by definition

$\Psi \vdash \llbracket \tau'\{M/x\} \rrbracket = \llbracket \tau' \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} :: \Pi t : \llbracket K\{M/x\} \rrbracket . \llbracket K'\{M/x\} \rrbracket$ by i.h.

$\Psi \vdash \llbracket \sigma\{M/x\} \rrbracket = \llbracket \sigma \rrbracket \{\{\llbracket M \rrbracket_c\}/x\} :: \llbracket K\{M/x\} \rrbracket$ by i.h.

Remaining cases are identical.

Lemma C.3 (Preservation of Equality).

1. *If* $\Psi \vdash K_1 = K_2$ *then* $\{\llbracket \Psi \rrbracket\} \vdash \llbracket K_1 \rrbracket = \llbracket K_2 \rrbracket$
2. *If* $\Psi \vdash \tau_1 = \tau_2 :: K$ *then* $\{\llbracket \Psi \rrbracket\} \vdash \llbracket \tau_1 \rrbracket = \llbracket \tau_2 \rrbracket :: \llbracket K \rrbracket$
3. *If* $\Psi \vdash A = B :: K$ *then* $\{\llbracket \Psi \rrbracket\} \vdash \llbracket A \rrbracket = \llbracket B \rrbracket :: \llbracket K \rrbracket$
4. *If* $\Psi \vdash M = N : \tau$ *then* $\{\llbracket \Psi \rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_z = \llbracket N \rrbracket_z :: z{:}\llbracket \tau \rrbracket$
5. *If* $\Psi; \Gamma; \Delta \vdash P = Q :: z{:}A$ *then* $\{\llbracket \Psi \rrbracket\}; \llbracket \Gamma \rrbracket; \llbracket \Delta \rrbracket \vdash \llbracket P \rrbracket = \llbracket Q \rrbracket :: z{:}\llbracket A \rrbracket$

Proof. By induction on the given judgment.

Case: KEqR, KEqS, KEqT and $\mathsf{KEq}\Pi_2$
Immediate by i.h.

Case: $\mathsf{KEq}\Pi_1$

$\Psi \vdash \tau = \sigma :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash K_1 = K_2$ by inversion

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket \tau \rrbracket = \llbracket \sigma \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \{\llbracket \tau \rrbracket\} = \{\llbracket \sigma \rrbracket\} :: \mathsf{type}$ by $\mathsf{TEq}\{\}$ $\{\llbracket \Psi \rrbracket\}, x{:}\{\llbracket \tau \rrbracket\} \vdash \llbracket K_1 \rrbracket = \llbracket K_2 \rrbracket$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \Pi x{:}\{\llbracket \tau \rrbracket\} . \llbracket K_1 \rrbracket = \Pi x{:}\{\llbracket \sigma \rrbracket\} . \llbracket K_2 \rrbracket$ by $\mathsf{KEq}\Pi_1$

(2)

Case: TEqR, TEqT, TEqS
Immediate by i.h.

Case: $\mathsf{TEq}\Pi$

$\Psi \vdash \tau = \tau' :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash \sigma = \sigma' :: \mathsf{type}$ by inversion

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket \tau \rrbracket = \llbracket \tau' \rrbracket :: \mathsf{stype}$ by i.h. $\{\llbracket \Psi \rrbracket\} \vdash \{\llbracket \tau \rrbracket\} = \{\llbracket \tau' \rrbracket\} :: \mathsf{type}$ by $\mathsf{TEq}\{\}$

$\{\llbracket \Psi \rrbracket\}, x{:}\{\llbracket \tau \rrbracket\} \vdash \llbracket \sigma \rrbracket = \llbracket \sigma' \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \forall x{:}\{\llbracket \tau \rrbracket\} . \llbracket \sigma \rrbracket = \forall x{:}\{\llbracket \tau' \rrbracket\} . \llbracket \sigma' \rrbracket :: \mathsf{stype}$ by $\mathsf{STEq}\forall$

Case: $\mathsf{TEq}\lambda$

$\varPsi \vdash \tau = \tau' :: \mathsf{type}$ and $\varPsi, x{:}\tau \vdash \sigma = \sigma' :: K$ by inversion
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket = \llbracket\tau'\rrbracket :: \mathsf{stype}$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \{\llbracket\tau\rrbracket\} = \{\llbracket\tau'\rrbracket\} :: \mathsf{type}$ by $\mathsf{TEq}\{\}$
$\{\llbracket\varPsi\rrbracket\}, x{:}\{\llbracket\tau\rrbracket\} \vdash \llbracket\sigma\rrbracket = \llbracket\sigma'\rrbracket :: \llbracket K\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \lambda x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket = \lambda x{:}\{\llbracket\tau'\rrbracket\}.\llbracket\sigma'\rrbracket :: \varPi x{:}\{\llbracket\tau\rrbracket\}.\llbracket K\rrbracket$ by $\mathsf{STEq}\lambda$

Case: $\mathsf{TEq}T\lambda$

$\varPsi \vdash K = K'$ and $\varPsi, t :: K \vdash \tau = \sigma :: K''$ by inversion
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket K\rrbracket = \llbracket K'\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\}, t :: \llbracket K\rrbracket \vdash \llbracket\tau\rrbracket = \llbracket\sigma\rrbracket :: \llbracket K''\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \lambda t :: \llbracket K\rrbracket.\llbracket\tau\rrbracket = \lambda t :: \llbracket K'\rrbracket.\llbracket\sigma\rrbracket :: \varPi t :: \llbracket K\rrbracket.\llbracket K''\rrbracket$ by $\mathsf{STEq}T\lambda$

Case: TEqApp

$\varPsi \vdash \tau = \sigma :: \varPi x{:}\tau'.K$ and $\varPsi \vdash M = N : \tau'$ by inversion
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket = \llbracket\sigma\rrbracket :: \varPi x{:}\{\llbracket\tau'\rrbracket\}.\llbracket K\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z = \llbracket N\rrbracket_z :: z{:}\llbracket\tau'\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \{\llbracket M\rrbracket_z\} = \{\llbracket N\rrbracket_z\} : \{\llbracket\tau'\rrbracket\}$ by $\mathsf{TEq}\{\}$
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket\,\{\llbracket M\rrbracket_z\} = \llbracket\sigma\rrbracket\,\{\llbracket N\rrbracket_z\} :: \llbracket K\rrbracket\{\{\llbracket M\rrbracket_z\}/x\}$ by $\mathsf{STEqApp}$
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket\,\{\llbracket M\rrbracket_z\} = \llbracket\sigma\rrbracket\,\{\llbracket N\rrbracket_z\} :: \llbracket K\{M/x\}\rrbracket$ by compositionality and conversion

Case: $\mathsf{TEqTApp}$

$\varPsi \vdash \tau = \tau' :: \varPi t :: K_1.K_2$ and $\varPsi \vdash \sigma = \sigma' :: K_1$ by inversion
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket = \llbracket\tau'\rrbracket :: \varPi t :: \llbracket K_1\rrbracket.\llbracket K_2\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\sigma\rrbracket = \llbracket\sigma'\rrbracket :: \llbracket K_1\rrbracket$ by i.h.
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket\,\llbracket\sigma\rrbracket = \llbracket\tau'\rrbracket\,\llbracket\sigma'\rrbracket :: \llbracket K_2\rrbracket\{\llbracket\sigma\rrbracket/t\}$ by $\mathsf{STEqTApp}$
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\tau\rrbracket\,\llbracket\sigma\rrbracket = \llbracket\tau'\rrbracket\,\llbracket\sigma'\rrbracket :: \llbracket K_2\{\sigma/t\}\rrbracket$ by compositionality and conversion

Case: $\mathsf{TEq}\beta$

$\varPsi, x{:}\tau \vdash \sigma :: K$ and $\varPsi \vdash M : \tau$ by inversion
$\{\llbracket\varPsi\rrbracket\}, x{:}\{\llbracket\tau\rrbracket\} \vdash \llbracket\sigma\rrbracket :: \llbracket K\rrbracket$ by type preservation of the encoding
$\{\llbracket\varPsi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_c :: c{:}\llbracket\tau\rrbracket$ by type preservation of the encoding
$\{\llbracket\varPsi\rrbracket\} \vdash \{\llbracket M\rrbracket_c\} : \{\llbracket\tau\rrbracket\}$ by $\{\}I$
$\{\llbracket\varPsi\rrbracket\} \vdash (\lambda x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket)\,\{\llbracket M\rrbracket_c\} = \llbracket\sigma\rrbracket\{\{\llbracket M\rrbracket_c\}/x\} :: \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by $\mathsf{STEq}\beta$
$\{\llbracket\varPsi\rrbracket\} \vdash (\lambda x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket)\,\{\llbracket M\rrbracket_c\} = \llbracket\sigma\{M/x\}\rrbracket :: \llbracket K\{M/x\}\rrbracket$
by compositionality and conversion

Case: $\mathsf{TEq}T\beta$

$\varPsi \vdash \sigma :: K$ and $\varPsi, t :: K \vdash \tau :: K'$ by inversion
$\{\llbracket\varPsi\rrbracket\} \vdash \llbracket\sigma\rrbracket :: \llbracket K\rrbracket$ by type preservation of the encoding
$\{\llbracket\varPsi\rrbracket\}, t :: \llbracket K\rrbracket \vdash \llbracket\tau\rrbracket :: \llbracket K'\rrbracket$ by type preservation of the encoding
$\{\llbracket\varPsi\rrbracket\} \vdash (\lambda t :: \llbracket K\rrbracket.\llbracket\tau\rrbracket)\,\llbracket\sigma\rrbracket = \llbracket\tau\rrbracket\{\llbracket\sigma\rrbracket/t\} :: \llbracket K'\rrbracket\{\llbracket\sigma\rrbracket/t\}$ by $\mathsf{STEq}T\beta$
$\{\llbracket\varPsi\rrbracket\} \vdash (\lambda t :: \llbracket K\rrbracket.\llbracket\tau\rrbracket)\,\llbracket\sigma\rrbracket = \llbracket\tau\{\sigma/t\}\rrbracket :: \llbracket K'\{\sigma/t\}\rrbracket$ by compositionality and conversion

Case: $\mathsf{TEq}\eta$

$\Psi \vdash \sigma :: \Pi x{:}\tau.K$ and $x \notin fv(\sigma)$ by inversion

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket \sigma \rrbracket :: \Pi x{:}\{\llbracket \tau \rrbracket\}.\llbracket K \rrbracket$ by type preservation of the encoding

$\{\llbracket \Psi \rrbracket\} \vdash \lambda x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket\, x = \llbracket \sigma \rrbracket :: \Pi x{:}\{\llbracket \tau \rrbracket\}.\llbracket K \rrbracket$ by $\mathsf{STEq}\eta$

$\{\llbracket \Psi \rrbracket\} \vdash \lambda x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket\, \{c \leftarrow (y \leftarrow x; [y \leftrightarrow c])\} = \llbracket \sigma \rrbracket :: \Pi x{:}\{\llbracket \tau \rrbracket\}.\llbracket K \rrbracket$
by STEqT, $\mathsf{STEqApp}$ and $\mathsf{TMEq}\{\}\eta$

Case: $\mathsf{TEqT}\eta$

$\Psi \vdash \tau :: \Pi t :: K_1.K_2$ and $t \notin fv(\tau)$ by inversion

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket \tau \rrbracket :: \Pi t :: \llbracket K_1 \rrbracket.\llbracket K_2 \rrbracket$ by type preservation of the encoding

$\{\llbracket \Psi \rrbracket\} \vdash \lambda t :: \llbracket K \rrbracket.\llbracket \tau \rrbracket\, t = \llbracket \tau \rrbracket :: \Pi t :: \llbracket K \rrbracket.\llbracket K' \rrbracket$ by $\mathsf{STEqT}\eta$

Case: $\mathsf{TEq}\{\}$

$\forall i, j.\Psi \vdash A_i = B_i :: \mathsf{stype}$, $\Psi \vdash C_j = D_j :: \mathsf{stype}$ and $\Psi \vdash A = B :: \mathsf{stype}$ by inversion

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket C_j \rrbracket = \llbracket D_j \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket A_i \rrbracket = \llbracket B_i \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \llbracket A \rrbracket = \llbracket B \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \overline{!\llbracket C_j \rrbracket} \multimap \overline{\llbracket A_i \rrbracket} \multimap \llbracket A \rrbracket = \overline{!\llbracket D_j \rrbracket} \multimap \overline{\llbracket B_i \rrbracket} \multimap \llbracket B \rrbracket :: \mathsf{stype}$ by $\mathsf{STEq} \multimap$ and $\mathsf{STEq!}$

(3)
All cases are identical to those of **(2)**.
(4)

Case: TMEqR

$\Psi \vdash M : \tau$ by inversion

$\{\llbracket \Psi \rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_z :: z{:}\llbracket \tau \rrbracket$ by type preservation of the encoding

$\{\llbracket \Psi \rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_z = \llbracket M \rrbracket_z :: z{:}\llbracket \tau \rrbracket$ by PEqR

Case: TMEqS TMEqT
Immediate by i.h. and the corresponding definitional equality rules for processes.

Case: $\mathsf{TMEq}\lambda$

$\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma$, $\Psi \vdash \lambda x{:}\tau'.N : \Pi x{:}\tau'.\sigma'$, $\Psi \vdash \Pi x{:}\tau.\sigma = \Pi x{:}\tau'.\sigma' :: \mathsf{type}$
and $\Psi, x{:}\tau \vdash M = N : \sigma$ by inversion

$\{\llbracket \Psi \rrbracket\}, x{:}\{\llbracket \tau \rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_z = \llbracket N \rrbracket_z :: z{:}\llbracket \sigma \rrbracket$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket = \forall x{:}\{\llbracket \tau' \rrbracket\}.\llbracket \sigma' \rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash z(x).\llbracket M \rrbracket_z = z(x').\llbracket N \rrbracket_z :: z{:}\forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket$ by $\mathsf{PEq}\forall\mathsf{R}$

Case: $\mathsf{TMEqApp}$

$\Psi \vdash M = M' : \Pi x{:}\tau.\sigma$ and $\Psi \vdash N = N' : \tau$ by inversion

$\{\llbracket \Psi \rrbracket\}; \cdot; \cdot \vdash \llbracket M \rrbracket_x = \llbracket M' \rrbracket_x :: x{:}\forall x{:}\{\llbracket \tau \rrbracket\}.\llbracket \sigma \rrbracket$ by i.h.

$\{\llbracket \Psi \rrbracket\}; \cdot; \cdot \vdash \llbracket N \rrbracket_y = \llbracket N' \rrbracket_y :: y{:}\llbracket \tau \rrbracket$ by i.h.

$\{\llbracket \Psi \rrbracket\} \vdash \{\llbracket N \rrbracket_y\} = \{\llbracket N' \rrbracket_y\} : \{\llbracket \tau \rrbracket\}$ by $\mathsf{TMEq}\{\}$

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} x)(\llbracket M\rrbracket_x \mid x\langle\{\llbracket N\rrbracket_y\}\rangle.[x \leftrightarrow z]) =$
$(\boldsymbol{\nu} x)(\llbracket M'\rrbracket_x \mid x\langle\{\llbracket N'\rrbracket_y\}\rangle.[x \leftrightarrow z]) :: z{:}\llbracket\sigma\rrbracket\{\{\llbracket N\rrbracket_y\}/x\}$ by PEqcut, PEq∀L, PEqID

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} x)(\llbracket M\rrbracket_x \mid x\langle\{\llbracket N\rrbracket_y\}\rangle.[x \leftrightarrow z]) =$
$(\boldsymbol{\nu} x)(\llbracket M'\rrbracket_x \mid x\langle\{\llbracket N'\rrbracket_y\}\rangle.[x \leftrightarrow z]) :: z{:}\llbracket\sigma\{N/x\}\rrbracket$ by compositionality and conversion

Case: TMEqβ

$\Psi \vdash \lambda x{:}\tau.M : \Pi x{:}\tau.\sigma$ and $\Psi \vdash N : \tau$ by inversion

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash y(x).\llbracket M\rrbracket_y :: y{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by type preservation of the encoding

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket N\rrbracket_w :: w{:}\llbracket\tau\rrbracket$ by type preservation of the encoding

$\{\llbracket\Psi\rrbracket\} \vdash \{\llbracket N\rrbracket_w\} : \{\llbracket\tau\rrbracket\}$ by $\{\}I$

To show: $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket(\lambda x{:}\tau.M)\, N\rrbracket_z = \llbracket M\{N/x\}\rrbracket_z :: z{:}\llbracket\sigma\{N/x\}\rrbracket$

S.T.S: $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} y)(y(x).\llbracket M\rrbracket_y \mid y\langle\{\llbracket N\rrbracket_w\}\rangle.[y \leftrightarrow z]) = \llbracket M\{N/x\}\rrbracket_z :: z{:}\llbracket\sigma\{N/x\}\rrbracket$

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} y)(y(x).\llbracket M\rrbracket_y \mid y\langle\{\llbracket N\rrbracket_w\}\rangle.[y \leftrightarrow z]) :: z{:}\llbracket\sigma\rrbracket\{\{\llbracket N\rrbracket_w\}/x\}$
by above, id,∀L and cut

$\rightarrow\rightarrow \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_w\}/x\}$ by operational semantics

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_w\}/x\} :: z{:}\llbracket\sigma\rrbracket\{\{\llbracket N\rrbracket_w\}/x\}$ by type preservation

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} y)(y(x).\llbracket M\rrbracket_y \mid y\langle\{\llbracket N\rrbracket_w\}\rangle.[y \leftrightarrow z]) = \llbracket M\{N/x\}\rrbracket_z :: z{:}\llbracket\sigma\{N/x\}\rrbracket$
by above, PEqRed, compositionality and conversion

Case: TMEqη

$\Psi \vdash M : \Pi x{:}\tau.\sigma$ and $x \notin fv(M)$ by inversion

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_y :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by type preservation of the encoding

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(x).(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid y\langle x\rangle.[y \leftrightarrow z]) =$
$z(x).(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid y\langle\{c \leftarrow (y \leftarrow x; [y \leftrightarrow c]) \leftarrow \cdot\}\rangle.[y \leftrightarrow z]) :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$
by PEqCut, PEq∀L, PEqID, TMEq$\{\}\eta$ and PEqR

To show: $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(x).(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid y\langle\{c \leftarrow (y \leftarrow x; [y \leftrightarrow c]) \leftarrow \cdot\}\rangle.[y \leftrightarrow z]) =$
$\llbracket M\rrbracket_z :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(x).(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid y\langle x\rangle.[y \leftrightarrow z]) = (\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid z(x).y\langle x\rangle.[y \leftrightarrow z]) :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$
by PEqCC∀

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid z(x).y\langle x\rangle.[y \leftrightarrow z]) = (\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid [y \leftrightarrow z]) :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$
by PEq$\forall\eta$

$(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid [y \leftrightarrow z]) \rightarrow \llbracket M\rrbracket_z$ by the operational semantics

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by type preservation

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid [y \leftrightarrow z]) = \llbracket M\rrbracket_z :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by PEqRed

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(x).(\boldsymbol{\nu} y)(\llbracket M\rrbracket_y \mid y\langle\{c \leftarrow (y \leftarrow x; [y \leftrightarrow c]) \leftarrow \cdot\}\rangle.[y \leftrightarrow z]) = \llbracket M\rrbracket_z :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$
by the above reasoning and PEqT

Case: TMEq$\{\}$

$\Psi; \overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash P = Q :: c{:}A$ by inversion

$\{\llbracket\Psi\rrbracket\}; \overline{u_j{:}\llbracket B_j\rrbracket}; \overline{d_i{:}\llbracket A_i\rrbracket} \vdash \llbracket P\rrbracket = \llbracket Q\rrbracket :: c{:}\llbracket A\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash c(u_0).\dots.c(u_j).c(d_0).\dots.c(d_n).\llbracket P\rrbracket =$
$c(u_0).\dots.c(u_j).c(d_0).\dots.c(d_n).\llbracket Q\rrbracket :: c{:}!\overline{\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$ by PEq⊸R, PEq!L

Case: $\mathsf{TMEq}\{\}\eta$

$\Psi \vdash M : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}$ by inversion
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z :: \overline{!\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$
To show: $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash c(u_0).\dots.c(u_j).c(d_0).\dots.c(d_n).\llbracket z \leftarrow M; \overline{u_j}; \overline{d_i}; [z \leftrightarrow c]\rrbracket$
$= \llbracket M\rrbracket_c :: c{:}\overline{!\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$
S.T.S: $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash c(u_0).\dots.c(u_j).c(d_0).\dots.c(d_n).(\boldsymbol{\nu} z)(\llbracket M\rrbracket_z \mid$
$\overline{z}\langle v_1\rangle.(\overline{u_1}\langle a_1\rangle.[a_1 \leftrightarrow v_1] \mid \cdots \mid \overline{z}\langle d_1\rangle.([y_1 \leftrightarrow d_1] \mid \cdots \mid \overline{z}\langle d_n\rangle.([y_n \leftrightarrow d_n] \mid [z \leftrightarrow c])\dots)$
$= \llbracket M\rrbracket_c :: c{:}\overline{!\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$
by $\mathsf{PEqCC} \multimap$ and $\mathsf{PEqCC!}$ and $\mathsf{PEq} \multimap \eta$ and $\mathsf{PEq!}\eta$, PEqR and PEqRed

Case: PEqR

$\Psi; \Gamma; \Delta \vdash P :: z{:}A$ by inversion
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P\rrbracket :: z{:}\llbracket A\rrbracket$ by type preservation of the encoding
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P\rrbracket = \llbracket P\rrbracket :: z{:}\llbracket A\rrbracket$ by PEqR

Case: PEqS and PEqT
Straightforward by i.h.
Case: PEqRed

$\Psi; \Gamma; \Delta \vdash P :: z{:}A$, $P \to^* Q$ and $\Psi; \Gamma; \Delta \vdash Q :: z{:}A$ by inversion
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P\rrbracket :: z{:}\llbracket A\rrbracket$ by type preservation of the encoding
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket Q\rrbracket :: z{:}\llbracket A\rrbracket$ by type preservation of the encoding
$\llbracket P\rrbracket \to^* \llbracket Q\rrbracket$ by operational correspondence
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P\rrbracket = \llbracket Q\rrbracket :: z{:}\llbracket A\rrbracket$ by PEqRed

Case: $\mathsf{PEq}\{\}E$

$\Psi \vdash M = N : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}$, $\Psi; \Gamma; \Delta, c{:}A \vdash Q = Q' :: z{:}C$, $\overline{u_j{:}B_j} \subseteq \Gamma$ and $\overline{d_i{:}A_i} = \Delta'$
by inversion
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_y = \llbracket N\rrbracket_y :: y{:}\overline{!\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$ by i.h.
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket, c{:}\llbracket A\rrbracket \vdash \llbracket Q\rrbracket = \llbracket Q'\rrbracket :: z{:}\llbracket C\rrbracket$ by i.h.
We conclude by PEqCut, (repeated) $\mathsf{PEq}{\multimap}\mathsf{L}$, $\mathsf{PEq!L}$ and PEqID.

All other process cases follow fairly straightforwardly by i.h.

Lemma C.4 (Preservation of Typing).

1. *If* $\Psi \vdash$ *then* $\llbracket\Psi\rrbracket \vdash$ *and* $\{\llbracket\Psi\rrbracket\} \vdash$.
2. *If* $\Psi \vdash K$ *then* $\{\llbracket\Psi\rrbracket\} \vdash \llbracket K\rrbracket$
3. *If* $\Psi \vdash \tau :: K$ *then* $\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau\rrbracket :: \llbracket K\rrbracket$
4. *If* $\Psi \vdash A :: K$ *then* $\{\llbracket\Psi\rrbracket\} \vdash \llbracket A\rrbracket :: \llbracket K\rrbracket$
5. *If* $\Psi \vdash M : \tau$ *then* $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z :: z{:}\llbracket\tau\rrbracket$
6. *If* $\Psi; \Gamma; \Delta \vdash P :: z{:}A$ *then* $\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P\rrbracket :: z{:}\llbracket A\rrbracket$

Proof. By induction on the given judgement. **(1)** is immediate by induction.

Case: $\tau = \Pi x{:}\tau'.\sigma$

$\Psi \vdash \tau' :: \mathsf{type}$ and $\Psi, x{:}\tau' \vdash \sigma :: \mathsf{type}$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket\Psi\rrbracket\}, x{:}\{\llbracket\tau'\rrbracket\} \vdash \llbracket\sigma\rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \forall x{:}\{\llbracket\tau'\rrbracket\}.\llbracket\sigma\rrbracket :: \mathsf{stype}$ by $\{\}$ and $\forall$ rules

Case: $\tau = \{\overline{u_j{:}B_j}; \overline{d_i{:}B_i} \vdash c{:}A\}$

Straightforward by induction.

Case: $\tau = \lambda x{:}\tau'.\sigma$

$\Psi \vdash \tau' :: \mathsf{type}$ and $\Psi, x{:}\tau' \vdash \sigma :: \mathsf{type}$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket\Psi\rrbracket\}, x{:}\{\llbracket\tau'\rrbracket\} \vdash \llbracket\sigma\rrbracket :: \mathsf{stype}$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \lambda x{:}\{\llbracket\tau'\rrbracket\}.\llbracket\sigma\rrbracket :: \mathsf{stype}$ by $\{\}$ and λ rules

Case: $\tau = \tau'\, M$

$\Psi \vdash \tau' :: \Pi x{:}\sigma.K$ and $\Psi \vdash M : \sigma$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \Pi x{:}\{\llbracket\sigma\rrbracket\}.\llbracket K\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_c :: c{:}\llbracket\sigma\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \{\llbracket M\rrbracket_c\} :: c{:}\{\llbracket\sigma\rrbracket\}$ by $\{\}I$

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket\, \{\llbracket M\rrbracket_c\} :: \llbracket K\rrbracket\{\{\llbracket M\rrbracket_c\}/x\}$ by application well-formedness rule

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket\, \{\llbracket M\rrbracket_c\} :: \llbracket K\{M/x\}\rrbracket$ by compositionality

Case: $\tau = \lambda t :: K.\tau'$

$\Psi, t :: K \vdash \tau' :: K_2$ by inversion

$\{\llbracket\Psi\rrbracket\}, t :: \llbracket K\rrbracket \vdash \llbracket\tau'\rrbracket :: \llbracket K_2\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \lambda t :: \llbracket K\rrbracket.\llbracket\tau'\rrbracket :: \Pi t :: \llbracket K\rrbracket.\llbracket K'\rrbracket$ by $T\lambda$ well-formedness rule

Case: $\tau = \tau'\, \sigma$

$\Psi \vdash \tau' :: \Pi t{::}K_1.K_2$ and $\Psi \vdash \sigma :: K_1$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \Pi t{::}\llbracket K_1\rrbracket.\llbracket K_2\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\sigma\rrbracket :: \llbracket K_1\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket\, \llbracket\sigma\rrbracket :: \llbracket K_2\rrbracket\{\llbracket K_1\rrbracket/t\}$ by $Tapp$ well-formedness rule

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket\, \llbracket\sigma\rrbracket :: \llbracket K_2\{K_1/t\}\rrbracket$ by compositionality

Case: $\tau = \tau'$ by conversion rule

$\Psi \vdash \tau' :: K$ by inversion

$\Psi \vdash K = K'$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \llbracket K\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket K\rrbracket = \llbracket K'\rrbracket$ by preservation of equality

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\tau'\rrbracket :: \llbracket K'\rrbracket$ by conversion rule

Case: $A = \mathbf{1}$

Immediate from the definition.

Case: $A = !A'$

Immediate by i.h and ! well-formedness rule.

Case: $A = A_1 \multimap A_2$

Immediate by i.h. and $\multimap$ well-formedness rule.

Case: $A = A_1 \otimes A_2$

Immediate by i.h. and $\otimes$ well-formedness rule.

Case: $A = \forall x{:}\tau.A_0$

$\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash A_0 :: \mathsf{stype}$	by inversion
$\{[\![\Psi]\!]\} \vdash [\![\tau]\!] :: \mathsf{stype}$	by i.h.
$\{[\![\Psi]\!]\}, x{:}\{[\![\tau]\!]\} \vdash [\![A_0]\!] :: \mathsf{stype}$	by i.h.
$\{[\![\Psi]\!]\} \vdash \forall x{:}\{[\![\tau]\!]\}.[\![A_0]\!] :: \mathsf{stype}$	by $\forall$ well-formedness rule

Case: $A = \exists x{:}\tau.A_0$

$\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash A_0 :: \mathsf{stype}$	by inversion
$\{[\![\Psi]\!]\} \vdash [\![\tau]\!] :: \mathsf{stype}$	by i.h.
$\{[\![\Psi]\!]\}, x{:}\{[\![\tau]\!]\} \vdash [\![A_0]\!] :: \mathsf{stype}$	by i.h.
$\{[\![\Psi]\!]\} \vdash \exists x{:}\{[\![\tau]\!]\}.[\![A_0]\!] :: \mathsf{stype}$	by $\exists$ well-formedness rule

Case: $A = \&\{\overline{l_i{:}B_i}\}$

Immediate by i.h. and $\&$ well-formedness rule.

Case: $A = \oplus\{\overline{l_i{:}B_i}\}$

Immediate by i.h. and $\oplus$ well-formedness rule.

Case: $A = \lambda x{:}\tau.A'$

$\Psi \vdash \tau :: \mathsf{type}$ and $\Psi, x{:}\tau \vdash A' :: K$	by inversion
$\{[\![\Psi]\!]\} \vdash [\![\tau]\!] :: \mathsf{stype}$	by i.h.
$\{[\![\Psi]\!]\}, x{:}\{[\![\tau]\!]\} \vdash [\![A']\!] :: [\![K]\!]$	by i.h.
$\{[\![\Psi]\!]\} \vdash \lambda x{:}\{[\![\tau]\!]\}.[\![A']\!] :: \Pi x{:}\{[\![\tau]\!]\}.[\![K]\!]$	by $S\lambda$ well-formedness rule
$\{[\![\Psi]\!]\} \vdash \lambda x{:}\{[\![\tau]\!]\}.[\![A']\!] :: [\![\Pi x{:}\tau.K]\!]$	by compositionality

Case: $A = A_0\, M$

$\Psi \vdash A_0 :: \Pi x{:}\tau.K$ and $\Psi \vdash M : \tau$	by inversion
$\{[\![\Psi]\!]\} \vdash [\![A_0]\!] :: \Pi x{:}\{[\![\tau]\!]\}.[\![K]\!]$	by i.h.
$\{[\![\Psi]\!]\}; \cdot; \cdot \vdash [\![M]\!]_c :: c{:}[\![\tau]\!]$	by i.h.
$\{[\![\Psi]\!]\} \vdash \{[\![M]\!]_c\} : \{[\![\tau]\!]\}$	by $\{\}$
$\{[\![\Psi]\!]\} \vdash [\![A_0]\!]\, \{[\![M]\!]_c\} :: [\![K]\!]\{\{[\![M]\!]_c\}/x\}$	by Sapp well-formedness rule
$\{[\![\Psi]\!]\} \vdash [\![A_0]\!]\, \{[\![M]\!]_c\} :: [\![K\{M/x\}]\!]$	by compositionality

Case: $A = \lambda t :: K.A'$

$\Psi, t :: K \vdash A' :: K_2$ and $\Psi \vdash K_1$	by inversion
$\{[\![\Psi]\!]\}, t :: [\![K]\!] \vdash [\![A']\!] :: [\![K_2]\!]$	by i.h.
$\{[\![\Psi]\!]\} \vdash [\![K_1]\!]$	by i.h.
$\{[\![\Psi]\!]\} \vdash \lambda t :: [\![K_1]\!].[\![A']\!] :: \Pi t :: [\![K_1]\!].[\![K_2]\!]$	by $S\Pi$ well-formedness rule
$\{[\![\Psi]\!]\} \vdash \lambda t :: [\![K_1]\!].[\![A']\!] :: [\![\Pi t :: K_1.K_2]\!]$	by compositionality

Case: $A = A'\,B$

$\Psi \vdash A' :: \mathit{\Pi} t :: K_1.K_2$ and $\Psi \vdash B :: K_1$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket A'\rrbracket :: \mathit{\Pi} t :: \llbracket K_1\rrbracket.\llbracket K_2\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket B\rrbracket :: \llbracket K_1\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket A'\rrbracket\,\llbracket B\rrbracket :: \llbracket K_2\rrbracket\{\llbracket B\rrbracket/x\}$ by S*app* well-formedness rule

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket A'\rrbracket\,\llbracket B\rrbracket :: \llbracket K_2\{B/x\}\rrbracket$ by compositionality

Case: $A = A'$ by conversion rule

$\Psi \vdash A' :: K$ and $\Psi \vdash K = K'$ by inversion

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket A'\rrbracket :: \llbracket K\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket K\rrbracket = \llbracket K'\rrbracket$ by preservation of equality

$\{\llbracket\Psi\rrbracket\} \vdash \llbracket A'\rrbracket :: \llbracket K'\rrbracket$ by conversion rule

Case: $M = \lambda x{:}\tau.M'$

$\Psi, x{:}\tau \vdash M : \sigma$ by inversion

$\{\llbracket\Psi\rrbracket\}, x{:}\{\llbracket\tau\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z :: z{:}\llbracket\sigma\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(x).\llbracket M\rrbracket_z :: z{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by $\forall\mathsf{R}$

Case: $M = M'\,N$

$\Psi \vdash M' : \mathit{\Pi} x{:}\tau.\sigma$ and $\Psi \vdash N : \tau$ by inversion

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M'\rrbracket_x :: x{:}\forall x{:}\{\llbracket\tau\rrbracket\}.\llbracket\sigma\rrbracket$ by i.h.

$\{\llbracket\tau\rrbracket\}; \cdot; \cdot \vdash \llbracket N\rrbracket_y :: y{:}\llbracket\tau\rrbracket$ by i.h.

$\{\llbracket\tau\rrbracket\} \vdash \{\llbracket N\rrbracket_y\} : \{\llbracket\tau\rrbracket\}$ by $\{\}I$

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} x)(\llbracket M'\rrbracket_x \mid x\langle\{\llbracket N\rrbracket_y\}\rangle.[x \leftrightarrow z]) :: z{:}\llbracket\sigma\rrbracket\{\{\llbracket N\rrbracket_y\}/x\}$ by cut, $\forall\mathsf{L}$ and id

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash (\boldsymbol{\nu} x)(\llbracket M'\rrbracket_x \mid x\langle\{\llbracket N\rrbracket_y\}\rangle.[x \leftrightarrow z]) :: z{:}\llbracket\sigma\{N/x\}\rrbracket$ by compositionality

Case: $M = x$

$\Psi, x{:}\tau \vdash x{:}\tau$ by assumption

$\{\llbracket\Psi\rrbracket\}, x{:}\{\llbracket\tau\rrbracket\}; \cdot; \cdot \vdash y \leftarrow x; [y \leftrightarrow z] :: z{:}\llbracket\tau\rrbracket$ by $\{\}E$ and id rules

Case: $M = \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\}$

$\Psi \vdash \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} : \{\overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash c{:}A\}$ by assumption

$\Psi; \overline{u_j{:}B_j}; \overline{d_i{:}A_i} \vdash P :: c{:}A$ by inversion

$\{\llbracket\Psi\rrbracket\}; \overline{u_j{:}\llbracket B_j\rrbracket}; \overline{d_i{:}\llbracket A_i\rrbracket} \vdash \llbracket P\rrbracket :: c{:}\llbracket A\rrbracket$ by i.h.

$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash z(u_0).\ldots.z(u_j).z(d_0).\ldots.z(d_n).\llbracket P\rrbracket :: z{:}!\overline{\llbracket B_j\rrbracket} \multimap \overline{\llbracket A_i\rrbracket} \multimap \llbracket A\rrbracket$
by $!\mathsf{L}$, copy and $\multimap\mathsf{R}$ (repeated)

Case: $M = M'$ by conversion rule

$\Psi \vdash M' : \sigma$ by inversion
$\Psi \vdash \sigma = \tau :: \mathsf{type}$ by inversion
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M'\rrbracket :: z{:}\llbracket\sigma\rrbracket$ by i.h.
$\{\llbracket\Psi\rrbracket\} \vdash \llbracket\sigma\rrbracket = \llbracket\tau\rrbracket :: \mathsf{stype}$ by preservation of equality
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M'\rrbracket :: z{:}\llbracket\tau\rrbracket$ by conversion rule

Case: $P = z\langle M\rangle.P_1$ by $\exists\mathsf{R}$

$\Psi \vdash M : \tau$ and $\Psi; \Gamma; \Delta \vdash P_1 :: z{:}A\{M/x\}$ by inversion
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_y :: y{:}\llbracket\tau\rrbracket$ by i.h.
$\{\llbracket\Psi\rrbracket\} \vdash \{\llbracket M\rrbracket_y\} : \{\llbracket\tau\rrbracket\}$ by $\{\}I$
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P_1\rrbracket :: z{:}\llbracket A\{M/x\}\rrbracket$ by i.h.
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash \llbracket P_1\rrbracket :: z{:}\llbracket A\rrbracket\{\{\llbracket M\rrbracket_y\}/x\}$ by compositionality
$\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash z\langle\{\llbracket M\rrbracket_y\}\rangle.\llbracket P_1\rrbracket :: z{:}\exists x{:}\{\llbracket\tau\rrbracket\}.\llbracket A\rrbracket$ by $\exists\mathsf{R}$

All other process cases follow straightforwardly by i.h. (and compositionality/preservation of equality when needed).

Theorem C.5 (Operational Correspondence – Completeness).

1. *Let $\Psi; \Gamma; \Delta \vdash P :: z{:}A$. If $P \to P'$ then $\llbracket P\rrbracket \to^+ Q$ with $\{\llbracket\Psi\rrbracket\}; \llbracket\Gamma\rrbracket; \llbracket\Delta\rrbracket \vdash Q = \llbracket P'\rrbracket :: z{:}A$*
2. *Let $\Psi \vdash M : \tau$. If $M \to M'$ then $\llbracket M\rrbracket_z \to N$ with $\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash N = \llbracket M'\rrbracket_z :: z{:}\llbracket\tau\rrbracket$*

Proof. By induction on the reduction relation.

Case: $(\boldsymbol{\nu}x)(x\langle M\rangle.P \mid x(y).Q) \to (\boldsymbol{\nu}x)(P \mid Q\{M/y\})$

$\llbracket(\boldsymbol{\nu}x)(x\langle M\rangle.P \mid x(y).Q)\rrbracket = (\boldsymbol{\nu}x)(x\langle\{\llbracket M_c\rrbracket\}\rangle.\llbracket P\rrbracket \mid x(y).\llbracket Q\rrbracket)$ by definition
$\to (\boldsymbol{\nu}x)(\llbracket P\rrbracket \mid \llbracket Q\rrbracket\{\{\llbracket M_c\rrbracket\}/y\})$ by operational semantics
$\llbracket(\boldsymbol{\nu}x)(P \mid Q\{M/y\})\rrbracket = (\boldsymbol{\nu}x)(\llbracket P\rrbracket \mid \llbracket Q\{M/y\}\rrbracket)$ by definition
$\Psi; \Gamma; \Delta \vdash (\boldsymbol{\nu}x)(\llbracket P\rrbracket \mid \llbracket Q\rrbracket\{\{\llbracket M_c\rrbracket\}/y\}) = (\boldsymbol{\nu}x)(\llbracket P\rrbracket \mid \llbracket Q\{M/y\}\rrbracket) :: z{:}C$
by compositionality, type preservation and PEqCut

Case: $c \leftarrow M \leftarrow \overline{u_j}; \overline{d_i}; Q \to c \leftarrow M' \leftarrow \overline{u_j}; \overline{d_i}; Q$ with $M \to M'$
Straightforward by i.h.
Case: $c \leftarrow \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} \leftarrow \overline{u_j}; \overline{d_i}; Q \to (\boldsymbol{\nu}c)(P \mid Q)$

$\llbracket c \leftarrow \{c \leftarrow P \leftarrow \overline{u_j}; \overline{d_i}\} \leftarrow \overline{u_j}; \overline{d_i}; Q\rrbracket \to^+ (\boldsymbol{\nu}c)(\llbracket P\rrbracket \mid \llbracket Q\rrbracket)$
by definition and operational semantics
$\llbracket(\boldsymbol{\nu}c)(P \mid Q)\rrbracket = (\boldsymbol{\nu}c)(\llbracket P\rrbracket \mid \llbracket Q\rrbracket)$ by definition
We conclude by PEqR.

Case: $(\lambda x{:}\tau.M)\,N \to M\{N/x\}$

$\llbracket(\lambda x{:}\tau.M)\,N\rrbracket_z = (\boldsymbol{\nu}y)(y(x).\llbracket M\rrbracket_y \mid y\langle\{\llbracket N\rrbracket_c\}\rangle.[y \leftrightarrow z])$ by definition
$\to (\boldsymbol{\nu}y)(\llbracket M\rrbracket_y\{\{\llbracket N\rrbracket_c\}/x\} \mid [y \leftrightarrow z]) \to \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\}$ by operational semantics
$\{\llbracket\Psi\rrbracket\}; \cdot; \cdot \vdash \llbracket M\rrbracket_z\{\{\llbracket N\rrbracket_c\}/x\} = \llbracket M\{N/x\}\rrbracket_z :: z{:}\llbracket\sigma\{N/x\}\rrbracket$
by compositionality and type preservation

Case: $M\,N \rightarrow M'\,N$ with $M \rightarrow M'$

$[\![M\,N]\!]_z = (\boldsymbol{\nu}x)([\![M]\!]_x \mid x\langle\{[\![N]\!]_c\}\rangle.[x \leftrightarrow z])$ by definition

$[\![M]\!]_x \rightarrow M_0$ with $\{[\![\Psi]\!]\}; \cdot; \cdot \vdash M_0 = [\![M']\!]_z :: z{:}A$ by i.h.

$(\boldsymbol{\nu}x)([\![M]\!]_x \mid x\langle\{[\![N]\!]_c\}\rangle.[x \leftrightarrow z]) \rightarrow (\boldsymbol{\nu}x)(M_0 \mid x\langle\{[\![N]\!]_c\}\rangle.[x \leftrightarrow z])$
by the operational semantics

$= (\boldsymbol{\nu}x)([\![M']\!]_x \mid x\langle\{[\![N]\!]_c\}\rangle.[x \leftrightarrow z]) :: z{:}A$ by type preservation, PEqCut and PEqR