Genericity and the π-Calculus

Martin Berger Kohei Honda Nobuko Yoshida

Abstract. Types in processes delineate specific classes of interactive behaviour in a compositional way. Key elements of process theory, in particular behavioural equivalences, are deeply affected by types, leading to applications in the descriptions and analyses of diverse forms of computing. As one of the examples of types for processes, this paper introduces a second-order polymorphic π-calculus based on duality principles, building on type structures coming from typed π-calculi, Linear Logic and game semantics. The calculus cleanly captures the main technical achievements of earlier work on second-order polymorphism, allowing a precise embedding of generic sequential functions as well as seamless integration with state and concurrency. In this paper, we study the linear polymorphic type discipline, the most basic form of duality-based polymorphism. Representing a class of strongly normalising polymorphic interaction, it allows us to illustrate various elements of the theory in a simple setting. A corresponding behavioural theory of polymorphic labelled transitions is studied, strengthening Pierce-Sangiorgi's polymorphic bisimulation via duality. As an application we give a fully abstract embedding of System F in the polymorphic π-calculus. The proof combines process-theoretic reasoning with techniques from game semantics. The abstract nature of our generic (polymorphic) labelled transitions plays an essential role, elucidating subtle aspects of polymorphism in functions and interaction.

1 Introduction

Types in processes delineate specific classes of interactive behaviour in a compositional way. Key elements of process theory, in particular behavioural equivalences, are deeply affected by types [25, 29, 36], allowing precise encapsulation of semantics of significant classes of computation [11, 21, 22, 48, 49]. If the theory of processes offers a basis for description of general computational behaviour [30], then types and processes could be a basis of description and analyses of diverse forms of the analysis of computational behaviour.

This paper studies one instance of types for processes, centring on the notion of genericity. Genericity is a useful concept in software engineering which allows encapsulation of design decisions such that data-structures and algorithms can be changed more independently. It arises in two distinct but closely related forms: one, which we may refer to as *universal*, aids generic manipulation of data, as in lists, queues, trees or stacks. The other *existential* form facilitates hiding of structure from the outside, asking for it to be treated generically. In both cases, genericity partitions programs into parts that depend on the precise nature of the data under manipulation and parts that do not, supporting principled code reuse and precise type-checking. For example, C++ evolved from C by adding genericity in the form of *templates* (universal) and *objects* (existential).

It is known that key aspects of genericity for sequential functional computation are captured by second-order polymorphism where type variables, in addition to program variables, can be abstracted and instantiated. In particular, the two forms of genericity mentioned above are accounted for by the two forms of quantification coming from logic, $\forall$ and $\exists$. Well-known formalisms incorporating genericity include System F (the second-order λ-calculus) [15, 43], ML [32] in its various guises and GJ (Generic Java) [14]. Centring on these and related formalisms, a rich body of studies on type disciplines, semantics and proof principles for genericity has been accumulated.

The present work aims to offer a π-calculus based starting point for repositioning and generalising the preceding functional account of genericity in the broader realm of interaction. We are partly motivated by the lack of a general mathematical basis of genericity that also covers state, concurrency and nondeterminism. For example, the status of two fundamental concepts for reasoning about generic computation, relational parametricity [43] and its dual simulation principle [1, 33, 42], is only well-understood for pure functions. But a mathematical basis of diverse forms of generic computation is important when we wish to reason about software made up from many components with distinct behavioural properties, from purely functional behaviour to programs with side effects to distributed computing, all of which may exhibit certain forms of genericity.

The π-calculus is a small syntax for communicating processes in which we can precisely represent many classes of computational behaviours [9, 11, 21, 22, 31, 48, 49]. Can we find a uniform account of genericity for diverse classes of computational behaviour using the π-calculus? This work presents our initial results in this direction, concentrating on a polymorphic variant of the linear π-calculus [48]. It turns out that the duality principle embodied in the linear type structure naturally extends to second-order quantification, leading to a powerful theory of polymorphism that allows precise embedding of existing polymorphic functional calculi and unifies some of the significant technical elements of the known theories of genericity.

Summary of Contributions. The present paper improves on the state of the art in several ways. We list the key technical challenges.

- Introduction of the linear polymorphic π-calculus based on duality principles. Here "linear" means, intuitively speaking, that processes always return an output when invoked. Operationally, linearity means processes always terminate. Proving our calculus to be linear is difficult and we resort to adapting methods based on reducibility candidates [2, 15, 16], in addition to process-theoretic ones [48], for showing strong normalisation (SN).
- A theory of behavioural equivalences based on a generic labelled transition system applicable to both sequential and concurrent polymorphic processes. Our transition relation is strongly guided by duality-based type information which contributes to the transition relation's high-level of abstraction, enhancing a construction by Pierce and Sangiorgi [37].

– To demonstrate the power of generic labelled transitions, we conclude with a fully abstract embedding of System F into the linear polymorphic π-calculus. The abstract nature of our generic transitions – in the sense of having comparatively few transitions – is most useful for obtaining full abstraction, because System F's impredicativity makes direct syntactic reasoning hard.

These constructions and results offer a firm basis upon which we can develop other elements of the theory including its extensions to more general behaviours including stateful and/or concurrent and non-terminating computation. The key nature of genericity in name passing interaction is found in a most acute form in the linear interaction, one of the most restricted classes of name passing interactions, both syntactically and semantically. We hope to clarify their impact on other realms of computation in our subsequent reports.

Related Work. Second-order polymorphism for the λ-calculus was developed by Girard [15] and Reynolds [43] who focused on universal abstraction. Later Mitchell and Plotkin [33, 42] relate the dual form, existential abstraction, to data hiding. Exploiting a duality principle, the present theory unifies these two uses of polymorphism, data-hiding and parametricty, into a single framework, both operationally and in typing. The unification is accompanied by new reasoning techniques such as generic labelled transition.

Turner [45] is the first to study (impredicative) polymorphism in the π-calculus, giving a type-preserving encoding of System F. His type discipline is incorporated into Pict [38]. Vasconcelos [46] studies a predicative polymorphic typing discipline and shows that it can type-check interesting polymorphic processes while allowing tractable type inference. Our use of a duality principle (whose origin can be traced back to Linear Logic [16]) is the main difference from those previous approaches.

Pierce and Sangiorgi [37] study a behavioural equivalence for Turner's calculus and observe that existential types can reduce the number of transitions by prohibiting interactions at hidden channels. Lazic, Nowak and Roscoe [26] show that when programs manipulate data abstractly (called *data independence*), a transition system with a parametricity property can be used for reasoning, leading to efficient model checking techniques. The generic labelled transition unifies, and in some cases strengthens, these ideas as dual aspects of a single framework.

Pitts studies contextual congruences in PCF-like polymorphic functional calculi and characterises them via syntactic logical relations [40, 41], cf. [39]. There is an analogous way to construct logical relations in the present setting, leading to a similar characterisation result (the construction is closely related with the duality-closed relation we use for the proof of strong normalisability). The transition-based reasoning discussed in the present paper gives a less abstract but more tractable reasoning technique.

Recently, several studies of the semantics of polymorphism based on games and other intensional models have appeared. Hughes [23] presents game semantics for polymorphism in which strategies pass arenas to represent type passing and proves full abstraction for System F. His model is somewhat complex due to its direct representation of type instantiation. Murawski and Ong [35] substantially simplify Hughes approach, but do not obtain full abstraction for impredicative polymorphism. Abramsky and Lenisa [5, 6] give a fully abstract model for predicative polymorphism using interaction combinators. [3] presents another model based on game semantics. Fully abstract embedding of impredicative polymorphism is left as an open issue in these works. In view of the relationship between π-calculi and game semantics [11, 20, 24], it would be interesting to use typed processes from the present work to construct game-based categories.

Structure of the paper. This paper is the first of a series detailing [12]. Its main purpose is to present the second-order linear typed π-calculus. The reader interested in related work is invited to consult [11, 21, 22, 48, 49]. Section 2 informally illustrates the key ideas with examples. Section 3 introduces the syntax and typing rules, and proves Subject Reduction. Section 4 establishes strong normalisability of linear polymorphic processes. Section 5 studies a generic labelled transition system and the induced equivalences. Section 6 demonstrates the reasoning power of generic labelled transitions by way of non-trivial examples. Section 7 concludes with a fully abstract embedding of System F.

Acknowledgement. Kohei Honda and Martin Berger are partially supported by EPSRC grant GR/S55545. Nobuko Yoshida is partially supported by EPSRC grants GR/R33465 and GR/S55538.

2 Generic Processes, Informally

This section introduces key ideas with simple examples. We start with the following small process which encodes the identity function, $\lambda y.y$, using the standard syntax of the (asynchronous) π-calculus [18].

$$!x(yz).\overline{z}\langle y\rangle$$

In this process $\overline{z}\langle y\rangle$ is an output of y along the channel z and $!x(yz).\overline{z}\langle y\rangle$ is a replicated input, repeatedly receiving two names y and z at x. After having received y and z, it sends y along z. For typing this process, we assign each channel by type which specifies how it can carry arguments. Let us assume the above identity function has functional type $\tau \Rightarrow \tau$. In the standard typing system of the π-calculus [29, 47], the above agent is typed as follows:

$$\vdash\ !x(yz).\overline{z}\langle y\rangle \triangleright x : (\tau(\tau))$$

where $(\tau(\tau))$ represents that name x inputs or outputs two arguments with type τ and type (τ). As first refinement, we attach *action modes* to types to ensure the linearity and direction of channel usage. The duality is defined between input and output. Hence x in the above identity agent is typed by:

$$(\overline{\tau}(\tau)^{\uparrow})^{!}$$

$\overline{\tau}$ indicates the dual of τ (which is inductively defined by dualising input and out modes). $(\tau)^{\uparrow}$ sends a name of type τ exactly once, while $(\overline{\tau}(\tau)^{\uparrow})^{!}$ indicates the behaviour of receiving two names at a replicated input channel, one used as $\overline{\tau}$ and the other as $(\tau)^{\uparrow}$.

Based on the linearity and duality, the polymorphic identity is now typed as follows.

$$\vdash !x(yz).\overline{z}\langle y\rangle \triangleright x : \forall \mathrm{X}.(\overline{\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}$$

Here X is a type variable and $\overline{\mathrm{X}}$ indicates the dual of X. $\forall \mathrm{X}$ universally abstracts X, saying X can be any type. It is important to bear in mind that $\forall \mathrm{X}$ binds X and its dual simultaneously (as in [16]). The operational content of typing a channel with a type variable is to enforce that y cannot be used as an interaction point (which would require a concrete type). Hence y with a variable $\overline{\mathrm{X}}$ only appears as a value in a message.

Next we consider the process which is dual to the above agent. Let $\mathsf{t}\langle y\rangle \stackrel{def}{=} !y(a_1a_2z).\overline{z}\langle a_1\rangle$, $\mathsf{not}\langle cw\rangle \stackrel{def}{=} !c(a_1a_2z).\overline{w}\langle a_2a_1z\rangle$ and $\mathbb{B} \stackrel{def}{=} \forall \mathrm{X}.(\overline{\mathrm{X}\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}$ (which are, respectively, truth, negation and the polymorphic boolean type).

$$\vdash \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \triangleright x : \exists \mathrm{X}.(\mathrm{X}(\overline{\mathrm{X}})^{\downarrow})^{?}, e : (\mathbb{B})^{\uparrow} \qquad (1)$$

This process sends y and z (respectively representing the truth and the continuation) via x, where $\overline{x}(yz)P$ stands for $(\boldsymbol{\nu}\, yz)(\overline{x}\langle yz\rangle|P)$. Then it receives a single name at z and sends its negation via e. $\downarrow$ means "input once" while $?$ stands for "output to a replicator". To understand the typing, let's look at the situation before existential abstraction:

$$\vdash \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \triangleright x : (\mathbb{B}(\overline{\mathbb{B}})^{\downarrow})^{?}, e : (\mathbb{B})^{\uparrow} \qquad (2)$$

We now abstract $\mathbb{B}$ and its dual at x simultaneously, obtaining $\exists \mathrm{X}.(\mathrm{X}(\overline{\mathrm{X}})^{\downarrow})^{?}$ ($\exists \mathrm{X}$ binds both X and $\overline{\mathrm{X}}$). Thus existential abstraction hides the concrete type $\mathbb{B}$.

The types $\forall \mathrm{X}.(\overline{\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}$ and $\exists \mathrm{X}.(\mathrm{X}(\overline{\mathrm{X}})^{\downarrow})^{?}$ are dual to each other and indicate that composition of two processes is possible. When composed, the process interacts as follows. Below and henceforth we write $\mathsf{id}\langle x\rangle$ for $!x(yz).\overline{z}\langle y\rangle$.

$$\begin{aligned}
\mathsf{id}\langle x\rangle \,|\, \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) &\longrightarrow \mathsf{id}\langle x\rangle \,|\, (\boldsymbol{\nu}\, yz)(\overline{z}\langle y\rangle | \mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \\
&\longrightarrow \mathsf{id}\langle x\rangle \;|\; (\boldsymbol{\nu}\, y)(\mathsf{t}\langle y\rangle | \overline{e}(c)\mathsf{not}\langle cy\rangle) \\
&\approx\;\; \mathsf{id}\langle x\rangle \;|\; \overline{e}(c)\mathsf{f}\langle c\rangle
\end{aligned}$$

Here $\mathsf{f}\langle c\rangle \stackrel{def}{=} !c(xyz).\overline{z}\langle y\rangle$ (representing falsity) and $\approx$ is the standard weak bisimilarity. As this interaction indicates, a universally abstracted name, after its receipt from the environment, can only be used to be sent back to the environment as a free name or be discarded for non-linear names. The dual existential side can then count on such behaviour of the interacting party: in the above case, the process on the right-hand side can expect that, via z, it would receive the name y as a free name which it has exported in the initial reduction, as it indeed does in the second transition.

This duality plays the key role in defining generic labelled transitions, which induce behavioural equivalences more abstract (larger) than non-generic ones and which are applicable to the reasoning over a wide range of generic behaviours. We use an example of a generic transition sequence of the process in (1).

$$\overline{x}(yz)(\mathsf{t}\langle y\rangle|z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \xrightarrow{\overline{x}(yz)}\xrightarrow{z\langle y\rangle} \mathsf{t}\langle y\rangle|\overline{e}(c)\mathsf{not}\langle cy\rangle \tag{3}$$

A crucial point in this transition is that it does *not* allow a bound input in the second action, because the protocol at existentially abstracted names is opaque. The induced name substitution then opens a channel for internal communication. In contrast, the process in (2), different from (1) only in type, has the following transition sequence.

$$\overline{x}(yz)(\mathsf{t}\langle y\rangle|z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \xrightarrow{\overline{x}(yz)}\xrightarrow{z(w)} \mathsf{t}\langle y\rangle|\overline{e}(c)\mathsf{not}\langle cw\rangle.$$

Note that we have a bound input in the second action; the transition sequence is now completely controlled by type information, without sending/receiving concrete values. Under this duality principle, quantified variables possible actions at w are no longer opaque but are specified by the type information $\mathbb{B}$. Here the duality principle dictates existential/universal type variables correspond to free name passing, while concrete types (which rigorously specify protocols of interaction by their type structure) correspond to bound name passing.

We further examine the effect of free input. The process in (3) has the following transitions next.

$$\mathsf{t}\langle y\rangle|\overline{e}(c)\mathsf{not}\langle cy\rangle \xrightarrow{\overline{e}(c)}\xrightarrow{c(w_1w_2c_1)}\xrightarrow{\tau}\xrightarrow{\overline{c_1}\langle w_2\rangle} \mathsf{t}\langle y\rangle|\mathsf{not}\langle cy\rangle \tag{4}$$

Here, because y is existentially quantified, $\mathsf{t}\langle y\rangle$ can never interact with processes in the environment. Hence an input transition at y never happens. Consequently, at the third step the only possible action is internal τ-action at y between $\mathsf{t}\langle y\rangle$ and message $\overline{y}\langle w_2w_1c_1\rangle$ generated from $\mathsf{not}\langle cy\rangle$. Thus the process in (2) in effect behaves as the falsity. On the other hand, in the process (4), $\mathsf{t}\langle y\rangle$ is always accessible from the environment, hence $\mathsf{t}\langle y\rangle$ and $\mathsf{not}\langle cw\rangle$ can behave independently without internal communication. For example, at the second step, it can have a bound input transition $y(w_1'w_2'c_1')$ from the environment.

This way, the duality in the type structure is precisely reflected in the duality in behaviour. This duality principle is also essential in the construction of the second-order semantic types for proving the strong normalisability of linear polymorphic processes and for various embedding results.

3 A Polymorphic π-Calculus

3.1 Processes

We use the asynchronous π-calculus [13, 18] to model genericity. In the light of Milner's encoding of the untyped λ-calculus, which contains various important calculi such as System F as subsyntax, it should not be surprising that there is no need to modify the asynchronous π-calculus.

For the readers' convenience, we briefly recapitulate key definitions. Let $x, y, \ldots$ range over a countable set $\mathcal{N}$ of *names*. Vectors of names are written $\vec{y}$. *Processes*, ranged over by $P, Q, R, \ldots$, are given by the following grammar.

$$P \quad ::= \quad x(\vec{y}).P \mid !x(\vec{y}).P \mid \overline{x}\langle\vec{y}\rangle \mid P|Q \mid (\boldsymbol{\nu}\, x)P \mid 0$$

Names in round parentheses act as binders, understood to be equipped with the usual α-equality $\equiv_\alpha$. $x(\vec{y}).P$ inputs via x with a continuation P. Its replicated counterpart is $!x(\vec{y}).P$. $\overline{x}\langle\vec{y}\rangle$ outputs $\vec{y}$ along x. In each of these agents, the leftmost free name the *subject*, while $\vec{y}$ is the vector of *objects*. The parallel composition of P and Q is $P|Q$ and $(\boldsymbol{\nu}\, x)P$ makes x private to P. 0 indicates the lack of behaviour. The structural equality $\equiv$ is the smallest congruence inductively generated from the following axioms.

- If $P \equiv_\alpha Q$ then $P \equiv Q$.
- $P|Q \equiv Q|P$, $P \equiv P|0$, $P|(Q|R) \equiv (P|Q)|R$, $P \equiv P|0$,
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$, $(\nu x)0 \equiv 0$, $x \notin \mathsf{fn}(P) \Rightarrow (\nu x)(P|Q) \equiv P|(\nu x)Q$.

The reduction relation $\longrightarrow$ is generated from:

$$\begin{array}{ll}
x(\vec{y}).P \,|\, \overline{x}\langle\vec{z}\rangle \;\longrightarrow\; P\{\vec{z}/\vec{y}\} & !x(\vec{y}).P \,|\, \overline{x}\langle\vec{z}\rangle \;\longrightarrow\; !x(\vec{y}).P \,|\, P\{\vec{z}/\vec{y}\} \\
P \longrightarrow Q \;\Rightarrow\; (\nu x)P \;\longrightarrow\; (\nu x)Q & P \longrightarrow P' \;\Rightarrow\; P|Q \;\longrightarrow\; P'|Q \\
P \equiv P' \longrightarrow Q' \equiv Q \;\Rightarrow\; P \longrightarrow Q &
\end{array}$$

The multi-step reduction $\twoheadrightarrow$ is defined as $(\longrightarrow \cup \equiv)^*$.

3.2 Types

Our typing judgements are of the form $\vdash_\phi P \triangleright A$ where the *action type* A summarises P's channel usage through *channels types* assigned to each of P's free names, and by causality arrows between the channel types. The *IO-mode* ϕ indicates whether P has an active thread or not and our linear types deal with three kinds of information.

- Directional: does P use the channel for input or output?
- Quantitative: how often is the channel being used?
- Qualitative: what kind of data flows over the channel?

In the following, we show how these ideas are defined and controlled by the linear type discipline.

Action Modes. Channel types are inductively made up from type variables and action modes. The four *action modes* speak about the directional and quantitative sides of channel usage:

$\downarrow$ Linear input,	$\uparrow$ Linear output,
$\mathbf{!}$ Server at replicated input,	$\mathbf{?}$ Client requests to $\mathbf{!}$.

Input modes are $\downarrow, \mathbf{!}$, while $\uparrow, \mathbf{?}$ are *output modes*. Input/output modes are together called *directed modes*. We let $p, p', \ldots$ (resp. p_{I}, resp. p_{O}) denote directed (resp. input, resp. output) modes. We define $\overline{p}$, the *dual* of p, by: $\overline{\downarrow} = \uparrow$, $\overline{\mathbf{!}} = \mathbf{?}$ and $\overline{\overline{p}} = p$.

Action modes may be best understood by considering what λ-abstractions do, seen from the outside: they receive an argument and may later return some value that (usually) depends on the argument. In our translation into the π-calculus (cf. Section 7), a λ-term is mapped to a process, and it just waits around for being invoked with an argument. This means it is a process $!x(\vec{v}).P$. Such a process is a *server* located at x. Replication is necessary, because a function, and hence its translation, may be used more than once, or not at all. Since everything is a process in π-calculi, arguments supplied to servers are the names of other servers, those where the argument is produced. When executing an invocation, a server asks the argument servers what their values are. Channels, like x in $!x(\vec{v}).P$, used by servers for invocations, have $\mathbf{!}$ as action mode. Dually, the client sends its invocation of a server on $\mathbf{?}$-moded channel. But what about the server returning a result to a client? We want the client to receive the result for its own unique last uncompleted invocation, hence the channel used for returning a value should be used at most once. If we know that the server will always terminate, as we do when translating strongly normalising calculi like System F, we can strengthen this requirement to channels for return values being used exactly once. That is the point of $\downarrow$ and (dually) $\uparrow$.

Syntax. We now formally define the syntax of channel types. *Channel types*, ranged over by $\tau, \sigma, \gamma, \ldots$, are given by the following grammar.

$$\tau ::= \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \mid \updownarrow \qquad \tau_{\mathtt{I}} ::= \mathrm{X} \mid \forall \vec{\mathrm{X}}.(\vec{\tau_{\mathtt{O}}})^{p_{\mathrm{I}}} \qquad \tau_{\mathtt{O}} ::= \mathrm{X} \mid \exists \mathrm{X}.(\vec{\tau_{\mathtt{I}}})^{p_{\mathrm{O}}}$$

where $\mathrm{X}, \mathrm{Y}, \ldots$ range over *type variables*, which is a countable set that comes with a self-inverse and irreflexive bijection $\overline{\mathrm{X}}$ (as found in [16]), i.e. $\overline{\overline{\mathrm{X}}} = \mathrm{X}$ and $\overline{\mathrm{X}} \neq \mathrm{X}$. Each X is assigned a directed action mode p, often written X^p, so that the mode of $\overline{\mathrm{X}}$ is always dual to that of X.

Often we have three different kinds of channel types: input channels, output channels and $\updownarrow$ which indicates that a channel is no longer available for *further* composition with the outside. In quantified types like $\forall \vec{\mathrm{X}}.(\vec{\tau_{\mathtt{O}}})^{p_{\mathrm{I}}}$, vectors $\vec{\mathrm{X}}$ and $\vec{\tau_{\mathtt{O}}}$ are not required to have the same or non-zero length. We write $(\vec{\tau})^{p_{\mathrm{I}}}$ to abbreviate $\forall \epsilon.(\vec{\tau})^{p_{\mathrm{I}}}$ and $(\vec{\tau})^{p_{\mathrm{O}}}$ is short for $\exists \epsilon.(\vec{\tau})^{p_{\mathrm{O}}}$. Quantifiers bind type variables in pairs, so that both X and $\overline{\mathrm{X}}$ in τ are bound in $\forall \mathrm{X}.\tau$ and $\exists \mathrm{X}.\tau$. One could say that the former is an abbreviation for $\forall \{\mathrm{X}, \overline{\mathrm{X}}\}.\tau$ and the latter for $\exists \{\mathrm{X}, \overline{\mathrm{X}}\}.\tau$. Type substitution, which should always respect action modes, is similarly shorthanded:

for example $(\overline{\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}\{\tau/\mathrm{X}\}$ is $(\overline{\tau}(\tau)^{\uparrow})^{!}$. The set $\mathsf{ftv}(\tau)$ of free type variables in τ also automatically includes duals, e.g. $\mathsf{ftv}(\mathrm{X}) = \{\mathrm{X}, \overline{\mathrm{X}}\}$. These formulation based on dual type variables follow [16].

A type τ is *closed* if $\mathsf{ftv}(\tau) = \emptyset$. $\tau_{\mathtt{I}}$ and $\tau_{\mathtt{O}}$ are called *input types* and *output types*, respectively. The *dual* $\overline{\tau}$ *of* τ, is the result of dualising all action modes, type variables and quantifiers in τ. For example $\overline{\forall \mathrm{X}.\exists \mathrm{Y}.((\mathrm{X})^{\downarrow}((\overline{\mathrm{Y}})^{\uparrow})^{!})^{?}} = \exists \mathrm{X}.\forall \mathrm{Y}.((\overline{\mathrm{X}})^{\uparrow}((\mathrm{Y})^{\downarrow})^{?})^{!}$. We set

$$\mathsf{md}(\mathrm{X}^p) = p, \quad \mathsf{md}((\vec{\tau})^p) = p, \quad \mathsf{md}(\forall \mathrm{X}.\tau) = \mathsf{md}(\exists \mathrm{X}.\tau) = \mathsf{md}(\tau), \quad \mathsf{md}(\updownarrow) = \updownarrow.$$

We often write τ^p if $\mathsf{md}(\tau) = p$. As in [11, 21, 48, 49], we assume the *sequentiality constraint* on channel types which requires that

- in $(\tau_1...\tau_n)^{!}$, $\mathsf{md}(\tau_j) = \uparrow$ for a unique j (for simplicity we assume $j = n$ from now on), else $\mathsf{md}(\tau_i) = \mathbf{?}$ and dually for $(\tau_1...\tau_n)^{?}$;
- in $(\tau_1...\tau_n)^{\downarrow}$, each τ_i has mode $\mathbf{?}$, dually for $(\tau_1...\tau_n)^{\uparrow}$.

These conditions (which come from [4, 20, 24]) ensure that names used for input carry only output names and vice versa (IO-alternation); further, a replicated input receives, besides arguments, exactly one name for answering to the client.

Composability of types is given by $\odot$ on channel types: $\odot$ is the least binary, associative and commutative partial operation on channel types such that

$$\tau^{\uparrow} \odot \overline{\tau}^{\downarrow} = \updownarrow \qquad \tau^{!} \odot \overline{\tau}^{?} = \tau^{!} \qquad \tau^{?} \odot \tau^{?} = \tau^{?}.$$

We will also write $\tau \asymp \sigma$ to indicate that $\tau \odot \sigma$ is defined. Intuitively, the first two rules say that a server should be unique, but an arbitrary number of clients can request interactions. The third rule says that once we compose input-output linear channels, the channel becomes uncomposable. Note that other compositions are undefined. For example, $!x(y).P \,|\, !x(z).Q$ is never typable because $(\tau)^{!} \odot (\tau)^{!}$ is undefined. $\overline{x} \mid x.\mathbf{0}$ is typable by $x : \updownarrow$, but $\overline{x} \mid \overline{x} \mid x.\mathbf{0}$ is not because $()^{\uparrow} \odot ()^{\uparrow}$ is undefined. This partial algebra of channel types ensures directional and quantitative information in typable processes by controlling their composability.

3.3 Typing

Sequentiality and Causality. The typing system proposed here adds quantifiers rules to the first-order system of [11, 48]. Here we introduce two key ideas, *sequentiality* and *causality of communications* for controlling qualitative aspects of process behaviours.

In sequents $\vdash_{\phi} P \triangleright A$ the IO-mode ϕ is an is either $\mathtt{I}$ or $\mathtt{O}$. The former, $\mathtt{I}$, indicates that P does not have an active output, i.e. an output that is not prefixed. The latter, $\mathtt{O}$, guarantees the existence of exactly one active output in P. In effect this removes concurrency from our typed calculus. This restriction can be lifted without affecting genericity but to keep things simple, we only treat the sequential case here. Formally we define the partial algebra:

$$\mathtt{I} \odot \mathtt{I} = \mathtt{I} \text{ and } \mathtt{I} \odot \mathtt{O} = \mathtt{O} \odot \mathtt{I} = \mathtt{O}.$$

Note $\mathtt{O} \odot \mathtt{O}$ is undefined, which means that we do not want more than one active thread at the same time. For example, $\overline{a} \mid \overline{a}$ and $b.(\overline{a} \mid \overline{c})$ are untypable by this constraint. We write $\phi_1 \asymp \phi_2$ if $\phi_1 \odot \phi_2$ is defined.

To ensure termination of process behaviour, we introduce the *action type* ranged over A, B, C..., which is a finite directed graph with nodes of the form $x\!:\!\tau$, such that

- no name occurs twice; and
- edges are of the form $x\!:\!\tau \to y\!:\!\tau'$ such that either (1) $\mathsf{md}(\tau) = \downarrow$ and $\mathsf{md}(\tau') = \uparrow$ or (2) $\mathsf{md}(\tau) = \mathbf{!}$ and $\mathsf{md}(\tau') = \mathbf{?}$.

Edges denotes dependency between channels and are used to prevent vicious cycles between names. $\mathsf{fn}(A)$ denote the set of names and modes in A and write $A(x)$ for the channel type assigned to x in A. We compose two processes typed by A and B when $A(a) \odot B(a)$ is defined for all $a \in \mathsf{fn}(A) \cap \mathsf{fn}(B)$, and a composition creates no circularity between names. For example, a composition of $x : \tau_1 \to y : \tau_2$ and $y : \overline{\tau_2} \to x : \overline{\tau_1}$ is undefined. We write $A \asymp B$ if $A \odot B$ is defined. The definition of $\odot$ is as the same as the first order case [48] and left to Appendix A. By this condition, for example, $a.\overline{b} \mid b.\overline{a}$ becomes untypable ([48] lists more examples). We note that $\odot$ on action types is a symmetric and associative partial operation with identity $\emptyset$, cf. [48].

(Zero)
$$\frac{-}{\vdash_{\mathtt{I}} \mathbf{0} \triangleright \emptyset}$$

(Par)
$$\frac{\vdash_{\phi_i} P_i \triangleright A_i \quad (i = 1, 2) \qquad A_1 \asymp A_2 \quad \phi_1 \asymp \phi_2}{\vdash_{\phi_1 \odot \phi_2} P_1 \mid P_2 \triangleright A_1 \odot A_2}$$

(Res)
$$\frac{\vdash_{\phi} P \triangleright A, x:\tau \qquad \mathsf{md}(\tau) \in \{\mathbf{!}, \updownarrow\}}{\vdash_{\phi} (\boldsymbol{\nu}\, x) P \triangleright A}$$

(Weak)
$$\frac{\mathsf{md}(\tau) \in \{\mathbf{?}, \updownarrow\} \qquad \vdash_{\phi} P \triangleright A^{\text{-}x}}{\vdash_{\phi} P \triangleright A, x:\tau}$$

(In$^{\downarrow}$) $(\mathrm{X}_i \notin \mathsf{ftv}(A, B))$
$$\frac{\vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}, \uparrow A^{\text{-}x}, \mathbf{?}B^{\text{-}x}}{\vdash_{\mathtt{I}} x(\vec{y}).P \triangleright (x : \forall \vec{\mathrm{X}}.(\vec{\tau})^{\downarrow} \to A), B}$$

(In$^{!}$) $(\mathrm{X}_i \notin \mathsf{ftv}(A))$
$$\frac{\vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}, \mathbf{?}A^{\text{-}x}}{\vdash_{\mathtt{I}} !x(\vec{y}).P \triangleright x : \forall \vec{\mathrm{X}}.(\vec{\tau})^{!} \to A}$$

(Out) $(\mathrm{X}_i \notin \mathsf{ftv}(\vec{\sigma}))$
$$\frac{\sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}\}}{\vdash_{\mathtt{O}} \overline{x}\langle \vec{y} \rangle \triangleright x : \exists \vec{\mathrm{X}}.(\vec{\tau})^{p_{\mathtt{O}}}, \vec{y} : \overline{\vec{\sigma}}}$$

Fig. 1. The polymorphic sequential typing rules.

Typing Rules. The typing rules are given in Figure 1. To simplify the presentation, we assume all newly introduced types to be well-formed. Some notation: $\mathsf{md}(A)$ denote the set of modes, $A/\vec{x}$ is the result of taking off nodes with names in $\vec{x}$ from A. A, B is the graph union of A and B with $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \emptyset$. $x\!:\!\tau \to A$ is a result of adding $x\!:\!\tau$ to A with an edge from $x\!:\!\tau$ to all of A's active nodes;

$A^{\text{-}x}$ for A such that $x \notin \mathsf{fn}(A)$; and $\vec{p}A$ indicates $\mathsf{md}(A) = \{\vec{p}\}$.

The (Zero) rule types 0, assigning it the $\mathtt{I}$-mode since there is no active output. As 0 has no free names, it is not being given any channel types. In (Par), $\phi_1 \asymp \phi_2$ ensures single threading because composition of two processes that are both $\mathtt{O}$-moded is prohibited. Similarly $A \asymp B$ guarantees the preservation of determinism and strong normalisation, in addition to consistent channel usage like linear inputs are only composed with linear outputs, etc. In (Res), we do not allow $\uparrow$, $\mathbf{?}$ or $\downarrow$-channels to be restricted since they carry actions which expect their dual actions to exist in the environment. (Weak) weakens with $\updownarrow$ and $\mathbf{?}$-nodes since these modes do not require *further* interaction. (In$^{\downarrow}$) ensures that x occurs precisely once (by $A^{\text{-}x}$) and no free input is suppressed under a prefix. In addition, this rule introduces universal quantification. The side conditions prevent capturing free type variables. This is in accord with the corresponding rules of the second-order λ-calculus. (In$^{\mathbf{!}}$) is the same as (In$^{\downarrow}$) except that no free $\uparrow$-channels are suppressed. This is because a $\uparrow$-channel under replication could be used more than once. Finally in (Out), we assume $z_i = z_j$ implies $\tau_i = \tau_j$ in $\vec{y}:\vec{\tau}$, where $\overline{\vec{\tau}}$ means pointwise dualisation of $\vec{\tau}$. This rule also adds existential quantification.

Simple examples of sequential polymorphic processes follow (expressions in the first three examples are from Section 2).

Example 1. 1. Let $\mathbb{I} \stackrel{def}{=} \forall \mathrm{X}.(\overline{\mathrm{X}}^{\mathbf{?}}(\mathrm{X}^{\mathbf{!}})^{\uparrow})^{\mathbf{!}}$. Then $\vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle \triangleright x:\mathbb{I}$.

2. $\vdash_{\mathtt{I}} \mathsf{t}\langle x\rangle \triangleright x:\mathbb{B}$, $\vdash_{\mathtt{I}} \mathsf{f}\langle x\rangle \triangleright x:\mathbb{B}$ and $\vdash_{\mathtt{I}} \mathsf{not}\langle xy\rangle \triangleright x:\mathbb{B}, y:\overline{\mathbb{B}}$.
3. We have $\vdash_{\mathtt{O}} \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle) \triangleright x:(\mathbb{B}(\overline{\mathbb{B}})^{\downarrow})^{\mathbf{?}}, e:(\mathbb{B})^{\uparrow}$.
4. Let $\mathsf{if}\ x\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \stackrel{def}{=} \overline{x}(b_1 b_2 z)(!b_1(\vec{v}a).P_1 | !b_2(\vec{v}a).P_2 | z(b).\overline{b}\langle\vec{v}a\rangle)$ We can check, with $\approx$ being the standard (untyped) weak bisimilarity:

$$\mathsf{if}\ x\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \,|\, \mathsf{t}\langle x\rangle \ \twoheadrightarrow\ P_1|\mathsf{t}\langle x\rangle|(\boldsymbol{\nu}\, b_2)!b_2(\vec{v}a).P_2 \ \approx\ P_1|\mathsf{t}\langle x\rangle$$

and symmetrically for $\mathsf{if}\ x\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \,|\, \mathsf{f}\langle x\rangle$. Assume $\vdash_{\mathtt{O}} P_{1,2} \triangleright \vec{v}:\vec{\tau}^{\mathbf{?}}, a:\tau^{\uparrow}$. Then we can type: $\vdash_{\mathtt{O}} \mathsf{if}\ x\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \triangleright x:\overline{\mathbb{B}}, \vec{v}:\vec{\tau}^{\mathbf{?}}, a:\tau^{\uparrow}$.
5. Let $\mathsf{fw}_{xy}^{n+1} \stackrel{def}{=} !x(a_1..a_n b).\overline{y}\langle a_1..a_n b\rangle$ be the *forwarder of arity* $n+1$. Then

$$\vdash_{\mathtt{I}} \mathsf{fw}_{xy}^{n+1} \triangleright x : \forall \vec{\mathrm{Y}}.(\overline{\tau}_1...\overline{\tau}_n\overline{\sigma})^{\mathbf{!}} \to y : \exists \vec{\mathrm{Z}}.(\tau_1...\tau_n\sigma)^{\mathbf{?}}$$

for arbitrary types $\vec{\tau}$ and σ, provided that $\mathsf{md}(\tau_i) = \mathbf{!}$ and $\mathsf{md}(\sigma) = \downarrow$. The vectors $\vec{\mathrm{Y}}$ and $\vec{\mathrm{Z}}$ are neither required to coincide in length nor depend on n. For most applications we will choose $\vec{\mathrm{Y}} = \vec{\mathrm{Z}} = \mathsf{ftv}(\vec{\tau}, \sigma)$. Then we can check $(\boldsymbol{\nu}\, w)(\mathsf{fw}_{xw}^{n+1} \mid \mathsf{fw}_{wy}^{n+1})$ is typable and has the same type as fw_{xy}^{n+1}, while $(\boldsymbol{\nu}\, w)(\mathsf{fw}_{xw}^{n+1} \mid \mathsf{fw}_{wx}^{n+1})$ is untypable because $(x:\tau \to w:\overline{\tau}) \not\asymp (w:\tau \to x:\overline{\tau})$.
6. The λ-calculus allows to implement the booleans in various ways [7, 8, 34], one of which is called the *Polymorphic Booleans.* Set

$$\mathsf{T} = \lambda t.\lambda f.t, \qquad \mathsf{F} = \lambda t.\lambda f.f, \qquad \mathsf{not} = \lambda b.\lambda t.\lambda f.(bf)t$$

In System F, the type of the two boolean values is $\mathbb{B}_\lambda = \forall \mathrm{X}.\mathrm{X} \Rightarrow (\mathrm{X} \Rightarrow \mathrm{X})$. These definitions can be translated into our calculus as follows (Section 7 will study the encoding in detail):

$$
\begin{aligned}
[\![\mathsf{T}]\!]_u &= \overline{u}(a)!a(y).\overline{y}(c)!c(tm).\overline{m}(d)!d(fn).\overline{n}\langle t\rangle \\
[\![\mathsf{F}]\!]_u &= \overline{u}(a)!a(y).\overline{y}(c)!c(tm).\overline{m}(d)!d(fn).\overline{n}\langle f\rangle \\
[\![\mathsf{Not}]\!]_u &= \overline{u}(a)!a(b).\overline{b}(c)!c(eg).\overline{g}(h)!h(tk).\overline{k}(l)!l(fr).(\boldsymbol{\nu}\, s)(\overline{e}\langle fs\rangle \mid s(v).\overline{v}\langle tr\rangle)
\end{aligned}
$$

In our calculus this will be mapped to $\mathbb{B}^\circ_\lambda \stackrel{def}{=} \forall \mathrm{X}.(((\overline{\mathrm{X}}^?((\overline{\mathrm{X}}^?(\mathrm{X})^\uparrow)^!)^\uparrow)^!)^\uparrow)^!$. Then we can easily derive

$$\vdash_{\mathsf{0}} [\![\mathsf{T}]\!]_u \triangleright u : (\mathbb{B}^\circ_\lambda)^\uparrow, \qquad \vdash_{\mathsf{0}} [\![\mathsf{F}]\!]_u \triangleright u : (\mathbb{B}^\circ_\lambda)^\uparrow, \text{ and } \vdash_{\mathsf{0}} [\![\mathsf{Not}]\!]_u \triangleright u : (\mathbb{B}^\circ_\lambda)^\uparrow.$$

Applying Not to T is also typable as:

$$\vdash_{\mathsf{0}} (\nu t)([\![\mathsf{T}]\!]_t \mid t(v).(\nu n)([\![\mathsf{Not}]\!]_n \mid n(w).\overline{v}\langle wu\rangle)) \triangleright u : (\mathbb{B}^\circ_\lambda)^\uparrow$$

We will later see that this last process is the translation of the System F application ($\mathsf{Not}\ \mathsf{T}$), located at u. It is contextually indistinguishable from $[\![\mathsf{F}]\!]_u$. $\mathbb{B}_\lambda$ can be generalised to $[n]_\lambda \stackrel{def}{=} \forall \mathrm{X}.(\underbrace{\mathrm{X} \Rightarrow \cdots \Rightarrow \mathrm{X}}_{n} \Rightarrow \mathrm{X})$. This type will later be translated to

$$\forall \mathrm{X}.(((\underbrace{\overline{\mathrm{X}}^?((\overline{\mathrm{X}}^?(\ldots(\overline{\mathrm{X}}^?}_{n}((\overline{\mathrm{X}}^?(\mathrm{X})^\uparrow)^!)^\uparrow)^!\ldots)^\uparrow)^!)^\uparrow)^!)^\uparrow)^!.$$

The n selectors are the located CBV translations of $\pi^n_i = \Lambda \mathrm{X}.\lambda x_1 \ldots x_n.x_i$:

$$[\![\pi^n_i]\!]_u = \overline{a}(a)!a(x_1 b_1)\ldots\overline{b_{n-1}}(a_n)!a_n(x_n r)\overline{r}\langle x_i\rangle.$$

Of course there is a simpler form of n-ary choice: $[n] \stackrel{def}{=} \forall \mathrm{X}.(\underbrace{\mathrm{X}\ldots\mathrm{X}}_{n}(\overline{\mathrm{X}})^\uparrow)^!$ with the selectors being of the form $x\langle y_1\ldots y_n r\rangle\overline{r}\langle y_i\rangle$. Sometime we don't want just a finite number of choices but rather infinitely many. The *Church Numerals* [7, 8, 34] are a way of representing the natural numbers without recourse to base types in pure calculi like System F: a number n is simply the n-fold application of a function f to some variable x. This is the unary representation of numbers. Polymorphism is needed to define the arithmetic operations. Church Numerals are of the form:

$$\mathbb{CN}(n) = \Lambda \mathrm{X}.\lambda f^{\mathrm{X}\Rightarrow\mathrm{X}}.\lambda x^{\mathrm{X}}.\underbrace{f\ldots f}_{n}\, x$$

and their type is $\mathbb{CN} = \forall \mathrm{X}.((\mathrm{X} \Rightarrow \mathrm{X}) \Rightarrow (\mathrm{X} \Rightarrow \mathrm{X}))$. A translations into the second-order π-calculus is given as:

$$\mathbb{CN}^\circ = \forall \mathrm{X}.((((\mathrm{X}^!(\overline{\mathrm{X}}^?)^\downarrow)^?((\overline{\mathrm{X}}^?(\mathrm{X}^!)^\uparrow)^!)^\uparrow)^!)^\uparrow)^!$$

We will later see that $\overline{u}(a)!a(b).\overline{b}(c)!c(fe).\overline{e}(g)!g(xh).\overline{f}\langle xh\rangle$ is the CBV translation of $\mathbb{CN}(1)$, located at u.

Remark 1. Unlike Turner's calculus [45], the present type discipline records existential quantifications and universal quantifications as mutually dual notions. This duality is central to the semantic and syntactic results in later sections. Apart from this, the operational content of type instantiation in the explicitly typed counterpart of the present system is essentially identical to Turner's one.

We make extensive use of the *Variable Convention* (VC) which says that all occurring free names in proof trees are different from all occurring bound names. The next lemma justifies the VCs.

Lemma 1. *1. Let σ be an injective renaming. Then, for any proof-tree T of $\vdash_\phi P \triangleright A$, $T\sigma$ is a proof-tree for $\vdash_\phi P\sigma \triangleright A\sigma$.*

2. Let σ be an injective renaming of type variables, respecting duality, i.e. $\overline{\sigma(\mathrm{X})} = \sigma(\overline{\mathrm{X}})$. Then, for any proof-tree T of $\vdash_\phi P \triangleright A$, $T\sigma$ is a proof-tree for $\vdash_\phi P \triangleright A\sigma$.

Proof. By induction on the shape of T. □

Proposition 1. *If $\vdash_\phi P \triangleright A$ and $P \equiv Q$, then $\vdash_\phi Q \triangleright A$.*

Proof. By induction on the derivation of $P \equiv Q$. See Lemma 27 in Appendix B. □

Lemma 2. *If $\vdash_\phi P \triangleright A$ then $\vdash_\phi P \triangleright A\{\vec{\gamma}/\vec{\mathrm{X}}\}$.*

Proof. By induction on the derivation of $\vdash_\phi P \triangleright A$. See Lemma 27 in Appendix B. □

Lemma 3. *Assume $\vdash_\phi P \triangleright A$ and $A\{\vec{x}/\vec{v}\}$ is defined. Then $\vdash_\phi P\{\vec{x}/\vec{v}\} \triangleright A\{\vec{x}/\vec{v}\}$.*

Proof. We shall establish if $A\{x/v\}$ is defined, then $\vdash_\phi P\{x/v\} \triangleright A\{x/v\}$ by induction on the derivation of $\vdash_\phi P \triangleright A$. See Lemma 29 in Appendix B. □

Lemma 4. *Let $\vec{v}$ and $\vec{y}$ be two tuples of names of the same length such that $i \neq j, y_i = y_j$ implies $\tau_i = \tau_j$. Assume furthermore that $\vec{v}$ and w are all fresh and distinct names.*

1. *Assume a type $\uparrow A, ?B, \vec{v} : \vec{\tau}^{?}$ Then $(A, B, \vec{v} : \vec{\tau})\{\vec{y}/\vec{v}\}$ is defined and equals $A, (B \odot \vec{y} : \vec{\tau})$.*
2. *Given the type $?A, \vec{v} : \vec{\tau}^{?}, w : \sigma^{\uparrow}$ Then $(A, \vec{v} : \vec{\tau}, w : \sigma)\{\vec{y}z/\vec{v}w\}$ is defined and equals $(A \odot \vec{y} : \vec{\tau}), z : \sigma$.*

Proof. By straightforward induction on the length of $\vec{v}$. □

Lemma 5. (1) *If $?A \asymp ?B$ and $x : \tau^{!} \asymp B$ then $x : \tau \to A \asymp B$.* (2) *$?A \asymp ?A$;* (3) *$?A \asymp C$, $?B \asymp C$ implies $A \odot B \asymp C$.*

Proof. Immediate from the definitions. □

Theorem 1. *(subject reduction) If $\vdash_\phi P \triangleright A$ and $P \longrightarrow Q$, then $\vdash_\phi Q \triangleright A$.*

Proof. By induction on the derivation of $P \longrightarrow Q$. There are two interesting cases. The first one is $\overline{x}\langle\vec{y}\rangle|x(\vec{v}).P \longrightarrow P\{\vec{y}/\vec{v}\}$. We proceed by induction on the derivation of the typing judgement. If the last applied typing rule was (Weak), the result follows immediately from the inner (IH), otherwise the inference was

$$\dfrac{\dfrac{\sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}\} \qquad \mathrm{X}_i \notin \mathsf{ftv}(\vec{\sigma})}{\vdash_{\mathsf{O}} \overline{x}\langle\vec{y}\rangle \triangleright x : \exists\vec{\mathrm{X}}.(\vec{\tau})^{\uparrow}, \vec{y} : \overline{\vec{\sigma}}}\,(\textsc{Out}) \qquad \dfrac{\vdash_{\mathsf{O}} P \triangleright \vec{v} : \overline{\vec{\tau}}, \uparrow A^{-x}, ?B^{-x} \quad \vec{\mathrm{X}} \notin \mathsf{ftv}(A,B)}{\vdash_{\mathsf{I}} x(\vec{v}).P \triangleright B, x : \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}})^{\downarrow} \to A}\,(\textsc{In}^{\downarrow})}{\vdash_{\mathsf{O}} \overline{x}\langle\vec{y}\rangle|x(\vec{v}).P \triangleright (x :\updownarrow, A, B) \odot \vec{y} : \overline{\vec{\sigma}} = x :\updownarrow, A, (B \odot \vec{y} : \overline{\vec{\sigma}})}\,(\textsc{Par})$$

where we assume

$$x : \exists\vec{\mathrm{X}}.(\overline{\vec{\tau}})^{\uparrow}, \vec{y} : \overline{\vec{\sigma}} \asymp B, x : \forall\vec{\mathrm{X}}.(\vec{\tau})^{\downarrow} \to A$$

as a hypothesis for the application of (Par). W.o.l.g. we can also assume (Weak) is not used between ($\textsc{In}^{\downarrow}$) and (Par) ((Weak) $\searrow_r$ (Par) by Lemma 25 in Appendix B). By the (VC) we can assume that $\vec{v}$ and $\vec{y}$ are disjoint. Since by (Out) $i \neq j, y_i = y_j$ implies $\tau_i^{?} = \tau_j^{?}$, we apply Lemma 4 (1) to see that $(\vec{v} : \overline{\vec{\tau}}, A, B)\{\vec{y}/\vec{v}\}$ is defined and equals $(A,B) \odot \vec{y} : \overline{\vec{\tau}}$ which is $A, (B \odot \vec{y} : \overline{\vec{\tau}})$. Hence by Lemma 3, $\vdash_{\mathsf{O}} P\{\vec{y}/\vec{v}\} \triangleright A, (B \odot \vec{y} : \overline{\vec{\tau}})$. Now Lemma 2 ensures that also $\vdash_{\mathsf{O}} P\{\vec{y}/\vec{v}\} \triangleright (A, (B \odot \vec{y} : \overline{\vec{\tau}}))\{\vec{\gamma}/\vec{\mathrm{X}}\}$. But $\{\vec{\mathrm{X}}\} \cap \mathsf{ftv}(A,B) = \emptyset$, hence we have $(A, (B \odot \vec{y} : \overline{\vec{\tau}}))\{\vec{\gamma}/\vec{\mathrm{X}}\} = A, (B \odot \vec{y} : \overline{\vec{\tau}}\{\vec{\gamma}/\vec{\mathrm{X}}\})$, so in fact $\vdash_{\mathsf{O}} P\{\vec{y}/\vec{v}\} \triangleright A, (B \odot \vec{y} : \overline{\vec{\sigma}})$. A final application of (Weak) on x establishes the result.

The second interesting case is much as the first, but more complicated: $\overline{x}\langle\vec{z}y\rangle|!x(\vec{w}v).P \longrightarrow P\{\vec{z}y/\vec{w}v\}|!x(\vec{w}v).P$. We proceed by induction on the derivation of $\vdash_{\mathsf{O}} \overline{x}\langle\vec{z}y\rangle|!x(\vec{w}v).P \triangleright A$. For usage of (Weak), the result follows immediately from the inner (IH), otherwise the inference was

$$\dfrac{\dfrac{\sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}\}, \sigma_y = \tau_y\{\vec{\gamma}/\vec{\mathrm{X}}\} \quad \mathrm{X}_i \notin \mathsf{ftv}(\vec{\sigma})}{\vdash_{\mathsf{O}} \overline{x}\langle\vec{z}y\rangle \triangleright x : \exists\vec{\mathrm{X}}.(\vec{\tau}\tau_y)^{?}, y : \overline{\sigma}_y^{\uparrow}, \vec{z} : \overline{\vec{\sigma}}^{?}}\,(\textsc{Out}) \qquad \dfrac{\vdash_{\mathsf{O}} P \triangleright v : \overline{\tau}_y, \vec{w} : \overline{\vec{\tau}}, ?A^{-x}}{\vdash_{\mathsf{I}} !x(\vec{w}v).P \triangleright x : \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}\overline{\tau}_y)^{!} \to A}\,(\textsc{In}^{!})}{\vdash_{\mathsf{O}} \overline{x}\langle\vec{z}y\rangle|!x(\vec{w}v).P \triangleright (x : \forall\vec{\mathrm{X}}.(\vec{\tau}\tau_y)^{!} \to A) \odot (y : \overline{\sigma}_y, \vec{z} : \overline{\vec{\sigma}})}\,(\textsc{Par})$$

By the (VC) we can assume that $\{\vec{z}, y\} \cap \{\vec{w}, v\} = \emptyset$. (Out) guarantees that $i \neq j, z_i = z_j$ implies $\tau_i^{?} = \tau_j^{?}$, which means we can apply Lemma 4 (2) to be ensured of $(v : \overline{\tau}_y, \vec{w} : \overline{\vec{\tau}}, A)\{y\vec{z}/v\vec{w}\}$'s being defined and equal to $y : \overline{\tau}_y, (\vec{z} : \overline{\vec{\tau}} \odot A)$. Hence by Lemma 3, Lemma 2 and $\vec{\mathrm{X}} \notin \mathsf{ftv}(A)$:

$$\vdash_{\mathsf{O}} P\{\vec{z}y/\vec{w}v\} \triangleright y : \overline{\sigma_y}, (\vec{z} : \overline{\vec{\tau}} \odot A).$$

By assumptions, we have: $\vec{z} : \overline{\vec{\sigma}} \asymp x : \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}\overline{\tau}_y)^{!} \to A$. Hence in particular $z : \overline{\vec{\sigma}} \asymp A$ and we can form $\vec{z} : \overline{\vec{\sigma}} \odot A$. By construction $x \notin \mathsf{fn}(A)$ and, as we don't have recursive types, $x \notin \{\vec{z}\}$. But then by Lemma 5: $(\vec{z} : \overline{\vec{\tau}} \odot A) \odot (x : \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}\overline{\tau}_y)^{!} \to A)$ is defined and equals $(x : \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}\overline{\tau}_y)^{!} \to A) \odot \vec{z} : \overline{\vec{\tau}}$. Adding the linear output thus means that $(y : \overline{\sigma_y}, (\vec{z} : \overline{\vec{\tau}} \odot A)) \odot (y : \overline{\sigma_y}, (\vec{z} : \overline{\vec{\tau}} \odot A)$ is defined and equal to $(x : \forall\vec{\mathrm{X}}.(\vec{\tau}\tau_y)^{!} \to A) \odot (y : \overline{\sigma}_y, \vec{z} : \overline{\vec{\sigma}})$. Thus we may compose

$$\vdash_{\mathsf{O}} P\{\vec{z}y/\vec{w}v\}|!x(\vec{w}v).P \triangleright (x : \forall\vec{\mathrm{X}}.(\tau_y\vec{\tau})^{!} \to A) \odot (y : \overline{\sigma}_y, \vec{z} : \overline{\vec{\sigma}})$$

as required. The remaining cases are all derived from the outer (IH) and Proposition 1. $\square$

4 Strong Normalisability

Strong normalisability (SN) is a significant property of the second-order λ-calculus, which is closely related to the parametric nature of its polymorphism [15]. This section shows that this property has a precise analogue in polymorphic linear processes. The proof extends the SN proof for the linear π-calculus in [48] (based on type-directed predicates combined with acyclicity of name usage) to the second order case. The extension uses ideas due to Girard [16] and Abramsky [2], constructing "reducibility candidates" [15] based on closure under double negation.

From the viewpoint of theories of types for processes, the following result and its accompanying proofs is interesting as a demonstration of how a duality-based type structure leads to a strong syntactic property of typed processes.

We first list basic properties of the reduction relation in typed processes. Below and henceforth we use the following notations.

- $P \Downarrow Q \stackrel{\text{def}}{\Leftrightarrow} P \longrightarrow^* Q \not\longrightarrow$.
- $P \Downarrow \stackrel{\text{def}}{\Leftrightarrow} \exists Q. P \Downarrow Q$. Further, $P \Uparrow \stackrel{\text{def}}{\Leftrightarrow} \forall n \in \mathbb{N}.\ P \longrightarrow^n$.
- $\mathsf{SN}(P) \stackrel{\text{def}}{\Leftrightarrow} \neg P \Uparrow$.
- $\mathsf{CSN}(P) \stackrel{\text{def}}{\Leftrightarrow} \mathsf{SN}(P) \wedge (P \Downarrow Q_{1,2} \Rightarrow Q_1 \equiv Q_2)$.

Proposition 2. (determinacy) *Let* $\vdash_\phi P \triangleright A$. *(1)* $P \longrightarrow P'$ *and* $\mathsf{SN}(P')$ *imply* $\mathsf{SN}(P)$. *(2)* $P \Downarrow Q_i$ *(*$i = 1, 2$*) imply* $Q_1 \equiv Q_2$. *(3)* $P \Downarrow \ \Leftrightarrow\ \mathsf{SN}(P)\ \Leftrightarrow\ \mathsf{CSN}(P)$.

Proof. We first note that if $P \longrightarrow Q_i$ $(i = 1, 2)$, then either $Q_1 \equiv Q_2$ or there exists R such that $Q_i \longrightarrow R$ $(i = 1, 2)$. Then the rest is standard following Proposition 2.2 in [48]. □

Another important property follows. The proof is as in [48].

Proposition 3. (acyclicity of names) *$G(P)$ denotes a directed graph such that; (1) nodes are* $\mathsf{fn}(P)$*; and (2) edges are given by:* $x \curvearrowright y$ *iff* $P \equiv (\boldsymbol{\nu}\, \vec{z})(Q|R)$ *such that* $Q \equiv x(\vec{w}).Q_0$ *or* $Q \equiv\ !x(\vec{w}).Q_0$ *where* $y \in \mathsf{fn}(Q_0)$, $x \notin \{\vec{z}\}$ *and* $y \notin \{\vec{z}\vec{w}\}$. *A* cycle *in $G(P)$ is a sequence of form* $x \curvearrowright y_1 ... \curvearrowright y_n \curvearrowright x$ $(n \geq 0)$ *with* $y_i \neq x$. *Then $G(P)$ has no cycle.*

The main theorem in this section follows.

Theorem 2. (strong normalisability) $\vdash P \triangleright A \ \Rightarrow\ \mathsf{CSN}(P)$.

We prove the above result following three steps. First we define the *extended reduction relation* $\mapsto$, which eliminates all *cuts* (mutually dual channels) in a typed process: since $\mapsto$ property includes $\longrightarrow$, the above theorem is an easy corollary from the termination of $\mapsto$. Next we define *semantic types* $\langle\!\langle A, \phi \rangle\!\rangle$ of type A and ϕ, which are sets of typed terms that converge when composed with all complementary processes. This part is divided into two stages. In the first stage we define a certain kind of "atomic" processes called *connected*, which cannot be decomposed into two other processes. In the second stage, we define the semantic types based on the reducible candidates over the connected processes. Finally in the main lemma, we prove that each typable connected process is in the corresponding semantic type. Since there exists a translation from a typable process into connected processes, termination of the former implies that of the latter, concluding the proof of Theorem 2.

4.1 Extended Reduction

In this subsection, we introduce the extended reduction $\mapsto$ and its key properties. The idea of $\mapsto$ is to capture known process-algebraic laws as one step reductions: We define $\mapsto$ as the typed compatible relations on typed processes modulo $\equiv$ which are generated by the following rules.

$$\begin{array}{ll} (\mathsf{E1}) & C[\overline{x}\langle\vec{v}\rangle]|x(\vec{y}).P \mapsto C[P\{\vec{v}/\vec{y}\}] \\ (\mathsf{E2}) & C[\overline{x}\langle\vec{v}\rangle]|!x(\vec{y}).P \mapsto C[P\{\vec{v}/\vec{y}\}] \,|\, !x(\vec{y}).P \\ (\mathsf{E3}) & (\boldsymbol{\nu}\, x)!x(\vec{y}).Q \mapsto 0 \end{array}$$

Here we assume the term on the left-hand side in each rule is well-typed and x is not bound by $C[\cdot]$. $\mapsto$ is called the *extended reduction* relation. A process is in *extended normal form* if it does not contain $\mapsto$-redex. $P \Downarrow_e$, $\mathsf{SN}_e(P)$ and $\mathsf{CSN}_e(P)$ are given as $P \Downarrow$, $\mathsf{SN}(P)$ and $\mathsf{CSN}(P)$ using $\mapsto$ instead of $\longrightarrow$. A $\mapsto$*-redex* is a pair of subterms which form a redex for $\mapsto$ in a given term.

As in Proposition 3.1 in [48], we can prove a subject reduction theorem w.r.t. $\mapsto$ and a CR property of $\mapsto$. In particular we note that:

Theorem 3. (subject reduction for $\mapsto$) *If $\vdash_\phi P \triangleright A$ and $P \mapsto Q$ then also $\vdash_\phi Q \triangleright A$.*

Proof. By induction on the derivation of $P \mapsto Q$ with a nested induction on the structure of $C[\cdot]$. The two non-trivial cases follow from Subject Reduction for $\longrightarrow$. □

Proposition 4. (determinacy) *Let all processes be typed below. If $P \mapsto P'$ and $\mathsf{SN}_e(P')$ then $\mathsf{SN}_e(P)$. Thus $P \Downarrow_e$ iff $\mathsf{SN}_e(P)$ iff $\mathsf{CSN}_e(P)$.*

We also list the key properties of the extended reduction which are used later.

Lemma 6. *Let $\vdash_\phi P \triangleright A$. In addition, for all statements below, apart from (3), we assume $P \not\mapsto$.*

1. *If $A = B, x : \updownarrow$ then $x \notin \mathsf{fn}(P)$ and $\vdash_\phi P \triangleright B$.*
2. *If for some x, $\mathsf{md}(A(x)) = \downarrow$, then $P \equiv Q|x(\vec{v}).R$. If $\mathsf{md}(A(x)) = \mathbf{!}$, then $P \equiv Q|!x(\vec{v}).R$.*
3. *If $\phi = \mathtt{I}$ and for all $x \in \mathsf{fn}(A)$ $\mathsf{md}(A(x)) \in \{\uparrow, \mathbf{?}\}$, then $P \equiv 0$.*
4. *If $\phi = \mathtt{O}$ and for all $x \in \mathsf{fn}(A)$: $\mathsf{md}(A(x)) \in \{\uparrow, \mathbf{?}\}$, then A is of the form $x : \exists\vec{\mathrm{X}}.(\vec{\tau})^{p_{\mathrm{O}}}, \vec{y} : \vec{\vec{\sigma}}, B$ where $\sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}\}$ and $\mathrm{X}_i \notin \mathsf{ftv}(\vec{\sigma})$. In addition, $P \equiv \overline{x}\langle\vec{y}\rangle$ and all the names in B are introduced by weakening.*

Proof. By straightforward induction on the derivation of $\vdash_\phi P \triangleright A$. □

Lemma 7. *Assume $\vdash_\phi P \triangleright A$ and $P \not\rightarrow$. Let If $\phi = \mathtt{I}$ and $A = x : \mathbb{B}$, then $P \equiv \mathsf{t}\langle x\rangle$ or $P \equiv \mathsf{f}\langle x\rangle$.*

Proof. By induction on the derivation of $\vdash_\phi P \triangleright A$. Below the numbers (1..4) correspond to those in Lemma 6. Clearly, neither (Zero) nor (Out) not (In$^\downarrow$) could have been applied last. If (Par) was used, it must have used $\vdash_\mathtt{I} Q \triangleright x : \mathbb{B}$ and $\vdash_\mathtt{I} R \triangleright B$ as premises. By (3) then $R \equiv 0$, so we can just use the (IH). If the last inference used (Res) to infer from $\vdash_\mathtt{I} Q \triangleright x : \mathbb{B}, a : \tau^!$, we know by (2) that $Q \equiv R|!a(\vec{v}).S$. Now $a \notin \mathsf{fn}(R)$, for otherwise $R|!a(\vec{v}).S \not\twoheadrightarrow$, contradicting our assumptions. But then $Q \equiv R|(\nu a)!a(\vec{v}).S \twoheadrightarrow R$, also a contradiction. Hence we must have restricted from $\vdash_\mathtt{I} Q \triangleright x : \mathbb{B}, a : \updownarrow$. But (1) tells us that $P \equiv Q|(\nu a)0$. Now we apply the (IH). If (Weak) was applied, we use the (IH). This leaves (In$^!$), where $\vdash_\mathtt{0} Q \triangleright t : \overline{\mathrm{X}}, f : \overline{\mathrm{X}}, f : (\mathrm{X})^\uparrow$ was the premise, then (4) guarantees that $Q \equiv \overline{r}\langle t\rangle$ or $Q \equiv \overline{r}\langle f\rangle$, completing the proof.

4.2 Connectedness

A basic syntactic notion in linear processes is *connectedness*, which is used extensively in the proofs in this and subsequent sections. In essence, connected processes are those which cannot be separated into two non-trivial connected processes, characterised via types as follows.

(A, ϕ) is *connected* if one of the following holds.

- $\phi = \mathtt{I}$ and either A contains a unique $\mathbf{!}$-node and zero or more $\mathbf{?}$-nodes, or A contains a unique $\downarrow$-node, a unique $\uparrow$-node and zero or more $\mathbf{?}$-nodes.
- $\phi = \mathtt{0}$ and A contains a unique $\uparrow$-node and zero or more $\mathbf{?}$-nodes.

If (A, ϕ) is connected, the unique $\uparrow/\mathbf{!}$ node of A is its *principal node*. Types of mode $\uparrow/\mathbf{!}$ are often called *principal types*. We call processes which have connected types *connected*.
By typing system, any input, replication and output are connected. As examples of parallel composition, $(!a.\overline{b} \mid !b.\overline{c})$ is not connected, but $(\boldsymbol{\nu}\, b)(!a.\overline{b} \mid !b.\overline{c})$ is. Similarly $a.\overline{b} \mid e.\overline{c}$ is not, but $(\boldsymbol{\nu}\, be)(a.\overline{b} \mid b.\overline{e} \mid e.\overline{c})$ is.

Connectedness has both practical and theoretical significance. First, in many practical examples including the embedding of programming languages, it is often enough to consider connected processes. Second, any process of an arbitrary action type can always be decomposed canonically into connected processes, so that results about connected processes easily extend to non-connected processes.

By replacing (Par)/(Weak) by (Par$_c$)/(Weak$_c$) below, and deleting (Res), we can generate all and only connected processes up to $\equiv$ as follows.

$$(\textsc{Par}_c)\quad \frac{\vdash_{\phi_i} P_i \triangleright A_i \quad \asymp_i A_i \quad \asymp_i \phi_i \quad (\odot_i A_i/\vec{y},\ \phi)\ \text{connected} \quad \mathsf{md}(\odot_i A_i(y_i)) \in \{\mathbf{!}, *\}}{\vdash_{\odot_i \phi_i} (\boldsymbol{\nu}\, \vec{y})\Pi_i P_i \triangleright (\odot_i A_i)/\vec{y}}
\qquad
(\textsc{Weak}_c)\quad \frac{\vdash_\mathtt{0} P \triangleright A^{\text{-}x} \quad \mathsf{md}(\tau) = \mathbf{?}}{\vdash_\mathtt{0} P \triangleright A, x : \tau}$$

where we use the following notations, assuming a family $\{A_i\}_{i\in I}$ ($I = \{1,2,\ldots,n\}$).

$$\odot_i A_i \stackrel{def}{=} \begin{cases} \emptyset & (I=\emptyset) \\ A_1 \odot A_2 \odot \cdots \odot A_n & (I \neq \emptyset, \text{if defined}) \end{cases}$$

$$\odot_i \phi_i \stackrel{def}{=} \begin{cases} \mathtt{I} & (I=\emptyset) \\ \phi_1 \odot \phi_2 \odot \cdots \odot \phi_n & (I \neq \emptyset, \text{if defined}) \end{cases}$$

$$\asymp_{i\in I} A_i = (\forall i \in I.\ \asymp_{j\in I\setminus\{i\}} A_j) \wedge (\forall i \in I.\ A_i \asymp \odot_{j\in I\setminus\{i\}} A_j)$$

$$\asymp_{i\in I} \phi_i = \bigwedge_{i\neq j}(\phi_i \asymp \phi_j)$$

By definition, $\asymp_{i\in I} A_i$ and $\asymp_{i\in I} \phi_i$ are always true when I is empty or a singleton. $\Pi_i P_i$ is the standard n-ary parallel composition. Note in the conclusion of ($\text{P}\small{\text{AR}}_{\mathsf{c}}$), shared names are immediately hidden by the restriction, to make the type connected.

Call the resulting system, *connected typing system.* We observe the following result (cf. Proposition 3.2 in [48]).

Proposition 5. *1. Let A, ϕ be connected. Then if $\vdash_\phi P \triangleright A$ is derivable by the original typing system, then for some $P \equiv P_0$, we have $\vdash_\phi P_0 \triangleright A$ in the connected typing system.*
2. If each process of each connected type $\Downarrow_e$-converges, then all typable processes $\Downarrow_e$-converge.

Proof. (1) is mechanical by rule induction in Figure 1 and the connected typing rules. For (2), we translate an arbitrary typable process into the bigger size of (a parallel composition of) connected processes (except $\mathbf{0}$, which is obvious by $\mathbf{0} \Downarrow_e$). We prove this by induction on typable terms. See Appendix C. □

4.3 Candidates

A *prime with type* $x{:}\tau$ (with τ an input or output type, similarly hereafter) is a typed process $\vdash_\phi P \triangleright A$ such that $A(x) = \tau$ and (A, ϕ) is connected. $P \Downarrow_e$ means $\exists Q.P \longmapsto^* Q \not\longmapsto$. $P_{\langle x:\tau\rangle}$ denotes a prime with type $x:\tau$. For a set $\mathcal{U}$ of primes with type $x{:}\tau$, we define:

$$\mathcal{U}^{\perp} \stackrel{def}{=} \{P_{\langle x:\overline{\tau}\rangle} \mid \forall Q \in \mathcal{U}.\, (\boldsymbol{\nu}\, x)(P|Q) \Downarrow_e \}.$$

In $P|Q$ above we assume $\mathsf{fn}(P) \cap \mathsf{fn}(Q) = \{x\}$ by appropriate renaming of other names, similarly in the rest of this section.

Proposition 6. (1) $\mathcal{U}^{\perp\perp} \supseteq \mathcal{U}$. (2) $\mathcal{U}_1 \subseteq \mathcal{U}_2$ *implies* $\mathcal{U}_1^{\perp} \supseteq \mathcal{U}_2^{\perp}$. (3) $\mathcal{U}^{\perp\perp\perp} = \mathcal{U}^{\perp}$. *Hence* $(\ \cdot\ )^{\perp\perp}$ *is idempotent.* (4) $\bigcap_i \mathcal{U}_i^{\perp} = (\bigcup_i \mathcal{U}_i)^{\perp}$. (5) $(\bigcap_i \mathcal{U}_i)^{\perp\perp} = (\bigcup_i \mathcal{U}_i^{\perp})^{\perp} = \bigcap_i \mathcal{U}_i^{\perp\perp}$.

Proof. Standard. (3) and (4) use (1) and (2). (5) is by (4). □

We are now ready to define a candidate. Below we write $\mathcal{U} \Downarrow_e$ when $P \Downarrow_e$ for each $P \in \mathcal{U}$. Let τ be a closed type such that $\mathsf{md}(\tau) \in \{\mathbf{!}, \uparrow\}$. Then a *candidate of type* τ is a set $\mathcal{U}$ of primes with type $x : \tau$ such that $\mathcal{U}^{\perp\perp} = \mathcal{U}$ and $\mathcal{U} \Downarrow_e$. Candidates are defined only for modes $\{\mathbf{!}, \uparrow\}$, which is enough by duality. By Proposition 6 (3), $\mathcal{U}^{\perp}$ is always a candidate for $\mathcal{U} \neq \emptyset$ of mode $\mathbf{?}$ or $\downarrow$. We note:

Proposition 7. *For each closed type with mode* $\mathbf{!}, \uparrow$*, there is a non-empty candidate of that type.*

Proof. Clearly $!x(\vec{y}).\overline{z}\langle\vec{y}\rangle \in \mathcal{C}$ is in the candidate with type $\forall \vec{\mathrm{X}}.(\vec{\tau})^{\mathbf{!}}$ and $\overline{x}\langle\vec{y}\rangle$ is in the candidate with type $\exists \vec{\mathrm{X}}.(\vec{\tau})^{\uparrow}$. □

$\mathcal{C}, \mathcal{C}', \ldots$ range over candidates. We write $\mathcal{C}_\tau$ for a family of candidates $\{\mathcal{C}_x\}_{x \in \mathcal{N}}$ with each $\mathcal{C}_x$ of type $x : \tau$, which are closed under injective renaming (i.e. if $\mathcal{C}_x, \mathcal{C}_y \in \mathcal{C}_\tau$ then $\mathcal{C}_y = \mathcal{C}_x\binom{xy}{yx}$ where $\binom{xy}{yx}$ permutes x and y). We call such a family an *abstract candidate*.

4.4 Typed Predicates for Termination

A *(typed) predicate of type* τ (τ closed), written $\mathcal{P} : \tau$, is a family $\{\mathcal{P}_x\}_{x \in \mathcal{N}}$, of which each $\mathcal{P}_x$ is a set of primes with type $x : \tau$, which are closed under injective renaming (as before). Note an abstract candidate is a special case of a typed predicate. We now define maps on typed predicates as follows.

(p-$\mathbf{!}$) $(\mathcal{P}_1 ... \mathcal{P}_n)^{\mathbf{!}}_x \stackrel{def}{=} \{P \mid Q_i \in \mathcal{P}_{i,y_i} \supset P \circ \overline{x}\langle y_1 ... y_n\rangle \circ Q_1 \circ .. \circ Q_{n-1} \in \mathcal{P}_{n,y_n}\}$
(p-$\uparrow$) $(\mathcal{P}_1 ... \mathcal{P}_n)^{\uparrow}_x \stackrel{def}{=} \{\overline{x}\langle y_1 ... y_n\rangle \circ Q_1 \circ ... \circ Q_n \mid Q_i \in \mathcal{P}_{i,y_i}\}^{\perp\perp}$
(p-$\forall$) $\forall \mathrm{X}.\mathcal{P}[\mathrm{X}]_x \stackrel{def}{=} \{P \mid \forall \mathcal{P}'.P \in \mathcal{P}[\mathcal{P}']_x\}$
(p-$\exists$) $\exists \mathrm{X}.\mathcal{P}[\mathrm{X}]_x \stackrel{def}{=} \{P \mid \exists \mathcal{P}'.P \in \mathcal{P}[\mathcal{P}']_x\}^{\perp\perp}$

We assume all mentioned processes, types and substitutions are well-typed. In (p-$\mathbf{!}$) and (p-$\uparrow$), if $\mathcal{P}_i$ is a predicate with type X, then $P \circ Q_i$ stands for P otherwise $(\boldsymbol{\nu}\, \mathsf{fn}(P) \cap \mathsf{fn}(Q))(P|Q)$. $\mathcal{P}'[\mathcal{P}]$ is the result of applying a function $\mathcal{P}'[\cdot]$ over typed predicates to $\mathcal{P}$: here $\mathcal{P}'[\ ]$ should be typed as (say) $\mathrm{X} \mapsto \tau$ so that if $\mathcal{P}$ is of type ρ then $\mathcal{P}'[\mathcal{P}]$ is of type $\tau\{\rho/\mathrm{X}\}$. A brief illustration of these rules:

- In $(\vec{\mathcal{P}})^{\mathbf{!}}$, we require any prime in the resulting predicate to become a "value" (i.e. a process with $\uparrow$-type) when composed with "resources" (i.e. processes with $\mathbf{?}$-type). In $(\mathcal{P})^{\uparrow}$, we construct a set of primes from components and close it under double negation. Note that by Proposition 7, there always exists Q_i such that $Q_i \in \mathcal{P}_{i,y_i}$, so that the relation is well-defined.
- In $\forall \mathrm{X}.\mathcal{P}[\mathrm{X}]$, we ask for convergence under all $\perp\perp$-closed predicates, i.e. candidates. The use of candidates is necessary since component predicates do not come from induction. Dually $\exists \mathrm{X}.\mathcal{P}[\mathrm{X}]$ requires convergence under some $\perp\perp$-closed predicate, and takes the $\perp\perp$-closure (cf. [40]) of the resulting processes.

Proposition 8. (1) *If each* $\mathcal{P}_i$ *is a candidate then* $(\vec{\mathcal{P}})^p$ *(*$p \in \{!, \uparrow\}$*) is a candidate.* (2) *If* $\mathcal{P}'[\mathcal{C}_\tau] : \tau'\{\tau/\mathrm{X}\}$ *is a candidate for each* $\mathcal{C}_\tau$ *with* $\mathsf{md}(\tau) = \mathsf{md}(\mathrm{X})$, *then both* $\forall\mathrm{X}.\mathcal{P}'$ *and* $\exists\mathrm{X}.\mathcal{P}'$ *are candidates.*

Proof. For $(\vec{\mathcal{P}}\mathcal{P}^\uparrow)^!$ we observe:

$$\begin{array}{rcl} P \in (\vec{\mathcal{P}}\mathcal{P})^! & \Leftrightarrow & \forall \vec{S} \in \vec{\mathcal{P}}_{\vec{y}}.\ P \circ \overline{x}\langle \vec{y}w\rangle \circ \vec{S} \in \mathcal{P} = \mathcal{P}^{\perp\perp} \\ & \Leftrightarrow & \forall \vec{S} \in \vec{\mathcal{P}}_{\vec{y}},\ T \in \mathcal{P}^\perp.\ P \circ \overline{x}\langle \vec{y}w\rangle \circ \vec{S} \circ T \Downarrow_e \\ & \Leftrightarrow & P \in \{\overline{x}\langle \vec{y}w\rangle \circ \vec{S} \circ T \mid \vec{S} \in \vec{\mathcal{P}}_{\vec{y}}, T \in \mathcal{P}^\perp\}^\perp \end{array}$$

hence done (in the second equivalence we use $\mathcal{P} = \mathcal{P}^{\perp\perp}$). For (p-$\forall$), we use Proposition 6(5) to derive

$$(\forall\mathrm{X}.\mathcal{P}')_x^{\perp\perp} = (\bigcap_{\mathcal{P}} \mathcal{P}'[\mathcal{P}^{\perp\perp}])_x^{\perp\perp} = (\bigcap_{\mathcal{P}} \mathcal{P}'[\mathcal{P}^{\perp\perp}]^{\perp\perp})_x = (\bigcap_{\mathcal{P}} \mathcal{P}'[\mathcal{P}^{\perp\perp}])_x = (\forall\mathrm{X}.\mathcal{P}')_x.$$

The remaining cases are immediate by definition. $\square$

Using these actions, we define predicates for termination at each (possibly open) type. Below we let $\mathsf{md}(\mathrm{X}) \in \{!, \uparrow\}$. ξ ranges over *environments*, which are functions from type variables to abstract candidates respecting modes.

$$\langle\!\langle (\tau_1^? .. \tau_n^? \rho^\uparrow)^! \rangle\!\rangle_\xi \stackrel{def}{=} (\langle\!\langle \overline{\tau_1} \rangle\!\rangle_\xi .. \langle\!\langle \overline{\tau_n} \rangle\!\rangle_\xi \langle\!\langle \rho \rangle\!\rangle_\xi)^! \qquad \langle\!\langle (\tau_1 .. \tau_n)^\uparrow \rangle\!\rangle_\xi \stackrel{def}{=} (\langle\!\langle \tau_1 \rangle\!\rangle_\xi .. \langle\!\langle \tau_n \rangle\!\rangle_\xi)^\uparrow$$
$$\langle\!\langle \mathrm{X} \rangle\!\rangle_\xi \stackrel{def}{=} \xi(\mathrm{X}) \quad \langle\!\langle \forall\mathrm{X}.\tau \rangle\!\rangle_\xi \stackrel{def}{=} \forall\mathrm{X}.(\lambda\mathcal{P}.\langle\!\langle \tau \rangle\!\rangle_{\xi\cdot\mathrm{X}\mapsto\mathcal{P}^{\perp\perp}}) \quad \langle\!\langle \exists\mathrm{X}.\tau \rangle\!\rangle_\xi \stackrel{def}{=} \exists\mathrm{X}.(\lambda\mathcal{P}.\langle\!\langle \tau \rangle\!\rangle_{\xi\cdot\mathrm{X}\mapsto\mathcal{P}^{\perp\perp}})$$

Proposition 9. (1) $\langle\!\langle \tau \rangle\!\rangle_\xi$ *is a candidate.* (2) $\langle\!\langle \tau\{\rho/\mathrm{X}\} \rangle\!\rangle_\xi = \langle\!\langle \tau \rangle\!\rangle_{\xi\cdot\mathrm{X}\mapsto\langle\!\langle \rho \rangle\!\rangle_\xi}$. (3) *If* $\mathrm{X} \notin \mathsf{ftv}(\tau)$, *then* $\langle\!\langle \tau \rangle\!\rangle_\xi = \langle\!\langle \tau \rangle\!\rangle_{\xi\cdot\mathrm{X}\mapsto\mathcal{C}}$.

Proof. All are proven by straightforward inductions on the structure of τ, using Proposition 8 for (1). $\square$

As an illustration of how $\langle\!\langle \tau \rangle\!\rangle_\xi$ works, let us verify $\mathsf{id}\langle x\rangle \in \langle\!\langle \mathbb{I} \rangle\!\rangle_x$. By (p-$\forall$) we have only to check $\mathsf{id}\langle x\rangle \in (\mathcal{C}(\mathcal{C})^\uparrow)_x^!$ for each $\mathcal{C}$. Take $Q \in \mathcal{C}_y$. Then $\mathsf{id}\langle x\rangle \circ \overline{x}\langle yw\rangle \circ Q \mapsto^2 \overline{w}(y)Q$. Since $\overline{w}(y)Q \in (\mathcal{C})^\uparrow$ by (p-$\uparrow$), we are done.

Finally we can interpret action types. By Proposition 5 (2), it is enough to consider their connected subset. Let (A, ψ) be connected with principal node $x{:}\tau$. Further let $\vec{y} \stackrel{def}{=} \mathsf{fn}(A)\backslash\{x\}$. Then we define:

$$\langle\!\langle A, \psi \rangle\!\rangle \stackrel{def}{=} \{\vdash_\psi P \triangleright A \mid \forall\xi.\ (\forall i.\ Q_i \in \langle\!\langle \overline{A(y_i)} \rangle\!\rangle_{\xi, y_i}) \supset (\boldsymbol{\nu}\,\vec{y})(P \mid \Pi_i Q_i) \in \langle\!\langle \tau \rangle\!\rangle_{\xi, x}\}.$$

where $\mathsf{fn}(Q_i) \cap \mathsf{fn}(Q_j) = \emptyset$ $(i \neq j)$ and $\mathsf{fn}(P) \cap \mathsf{fn}(Q_i) = \{y_i\}$ for each i.
By definition we have:

Lemma 8. *If* $P \in \langle\!\langle A, \phi \rangle\!\rangle$ *then* $P \Downarrow_e$.

A key lemma follows. The proof is given in Section 4.5.

Lemma 9. (main lemma) $\vdash_\phi P \triangleright A$ *with* (A, ϕ) *connected implies* $P \in \langle\!\langle A, \phi \rangle\!\rangle$.

By Church-Rosser of $\mapsto$, its convergence implies strong normalisability. Now Theorem 2 follows from Proposition 5 (2) and Lemmas 8 and 9.

4.5 Proof of the Main Lemma

To prove Lemma 9, we use the following properties. Below $=_e$ is the convertibility relation from $\mapsto$ (i.e. the symmetric and transitive closure of $\mapsto$).

Proposition 10.

1. If $P \mapsto P'$ and $P' \in \mathcal{C}$ then $P \in \mathcal{C}$.
2. *If $P|Q \Downarrow_e$ then $P \Downarrow_e$ and $Q \Downarrow_e$. Further if P and Q share only output subjects then $P|Q \Downarrow_e$ iff $P \Downarrow_e$ and $Q \Downarrow_e$. Similarly for $(\boldsymbol{\nu}\, x)P$, $!a(\vec{x}).R$ and $a(\vec{x}).R$.*
3. If $P =_e Q$ and $P \Downarrow_e$ then $Q \Downarrow_e$.
4. Let $\vdash_\phi P \triangleright A$ and $\vdash_{\mathtt{I}} R \triangleright x : \tau^{!} \to B^{\text{-}y}$ such that (1) $A(x) = A(y) = \overline{\tau}$ and (2) $\mathsf{fn}(A)/x \cap \mathsf{fn}(B) = \emptyset$. Then $(\boldsymbol{\nu}\, xy)(P|R|R\{y/x\}) =_e (\boldsymbol{\nu}\, x)(P\{x/y\}|R)$.

Proof. (1) and (3) are by Church-Rosser of $\mapsto$. (2) and (4) are by definition. □

We now prove Lemma 9 by induction on the size of connected processes. We divide two cases: first we prove the cases of type $(\vec{\tau})^p$, then prove the cases for $\forall \mathrm{X}.\tau$ and $\exists \mathrm{X}.\tau$. Throughout the following reasoning, we assume $A = z_1 : \tau_1^{?}, .., z_n : \tau_n^{?}$ and set $\langle\!\langle \overline{A} \rangle\!\rangle_\xi \stackrel{def}{=} \{\Pi_{i \neq n} Q_i | Q_i \in \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi, z_i}\}$, unless otherwise stated. (IH) stands for induction hypothesis. Below we can safely ignore weakening since it does not affect the convergence.

Case $x(\vec{y}).P$: Then by $(\mathrm{I}\textsc{n}^{\downarrow})$, we have: $\vdash_{\mathtt{I}} x(\vec{y}).P \triangleright x : (\vec{\tau})^{\downarrow} \to z : \rho^{\uparrow}, ?A$. Fix $R \in \langle\!\langle \overline{A} \rangle\!\rangle_\xi$ and let $Q \in \langle\!\langle \overline{(\vec{\tau})^{\downarrow}} \rangle\!\rangle_x$. By the definition of $\langle\!\langle (\overline{\vec{\tau}})^{\uparrow} \rangle\!\rangle_\xi$ we know $Q \Downarrow_e \overline{x}(\vec{y})Q' \in \langle\!\langle (\overline{\vec{\tau}})^{\uparrow} \rangle\!\rangle_{\xi,x}$ with $Q' \in \langle\!\langle \vec{y} : \overline{\vec{\tau}} \rangle\!\rangle$. By (IH) $(\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{\vec{y}\})(P \,|\, Q' \,|\, R) \in \langle\!\langle \rho \rangle\!\rangle_z$, hence by Proposition 10 (1) we have $(\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{x\})(x(\vec{y}).P \,|\, Q \,|\, R) \in \langle\!\langle \rho \rangle\!\rangle_z$, as required.

Case $!x(\vec{y}).P$: Let $\underline{\vdash_{\mathtt{I}} !x(\vec{y}).P} \triangleright x : (\tau_1..\tau_n^{\uparrow})^{!} \to ?B, C$ is inferred by $(\mathrm{I}\textsc{n}^{!})$. Fix $A = B, C$ and $Q \in \langle\!\langle \overline{\vec{y} \backslash y_n : \vec{\tau} \backslash \vec{\tau_n}} \rangle\!\rangle_\xi$. Let $R \in \langle\!\langle \overline{A} \rangle\!\rangle$. Starting from (IH):

$$
\begin{array}{lll}
(\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{\vec{y}\})(P|Q|R) \in \langle\!\langle \tau_n \rangle\!\rangle_{\xi, y_n} & & \\
\quad \Rightarrow\ (\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{x\})(!x(\vec{y}).P|\overline{x}\langle y_1..y_n \rangle|Q|R) \in \langle\!\langle \tau_n \rangle\!\rangle_{\xi, y_n} & \text{Prop. 10 (1)} \\
\quad \Rightarrow\ (\boldsymbol{\nu}\, \mathsf{fn}(A))(!x(\vec{y}).P|R) \in \langle\!\langle (\vec{\tau})^{!} \rangle\!\rangle_{\xi, x} & \text{Def of } \langle\!\langle (\vec{\tau})^{!} \rangle\!\rangle_{\xi, x}
\end{array}
$$

as required.

Case $\overline{x}\langle \vec{z} \rangle$ where type of x is $(\vec{\tau})^{\uparrow}$: Let $\vdash_{\mathtt{O}} \overline{x}\langle \vec{z} \rangle \triangleright A, x : (\vec{\tau})^{\uparrow}$ be derived by $(\mathrm{O}\textsc{ut})$. Then:

$$
Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, z_i} \quad \Rightarrow \quad (\boldsymbol{\nu} \vec{\ })(\overline{x}\langle \vec{z} \rangle \mid \Pi_i Q_i) \in \langle\!\langle (\vec{\tau})^{\uparrow} \rangle\!\rangle_{\xi, x} \ (\text{Def of } \langle\!\langle (\vec{\tau})^{\uparrow} \rangle\!\rangle_{\xi, x})
$$

as required.

Case $\overline{x}\langle \vec{y} \rangle$ where type of x is $(\vec{\tau})^{?}$. Since $\tau_i \neq \mathrm{X}$, we assume $\vdash_{\mathtt{O}} (\boldsymbol{\nu}\, \vec{y})(\overline{x}\langle \vec{y} \rangle \mid \Pi Q_i) \triangleright A, x : (\vec{\tau})^{?}$ is derived by $(\mathrm{O}\textsc{ut})$ and $(\mathrm{P}\textsc{ar}_c)$ with $A = (\odot_i A_i)$, $\vdash_{\mathtt{I}} Q_i \triangleright y_i : \tau_i, A_i$ and $x \notin \mathsf{fn}(A_i)$. Fix $\vec{\tau} = \tau_1..\tau_n^{\downarrow}$ and $A = z : \rho^{\uparrow}, ?B$. Note Q_n has a principal name z by typing. Now fix $R \in \langle\!\langle \overline{B} \rangle\!\rangle_\xi$ and $T \in \langle\!\langle (\overline{\vec{\tau}})^{!} \rangle\!\rangle_{\xi, x}$. By induction hypothesis we have:

$$
(\boldsymbol{\nu}\, \mathsf{fn}(A))(Q_i|R) \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i} \ (i \neq n) \tag{5}
$$

as well as, for any $T_0 \in \langle\!\langle \overline{\tau_n} \rangle\!\rangle_{\xi, y_n}$,

$$(\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{y_n\})(Q_n | R | T_0) \in \langle\!\langle \rho \rangle\!\rangle_{\xi, z}. \tag{6}$$

By typing, we know $T \Downarrow_e !x(\vec{y}).T'$. Let $Q'_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i}$ $(1 \leq i \leq n-1)$. By the definition of $\langle\!\langle (\overline{\vec{\tau}})^! \rangle\!\rangle_{\xi, x}$, we have:

$$(\boldsymbol{\nu}\, y_1..y_{n-1})(T' | \Pi_{1 \leq i \leq n-1} Q'_i) \in \langle\!\langle \overline{\tau_n} \rangle\!\rangle_{\xi, y_n} \tag{7}$$

We can now reason:

$$\begin{array}{lll}
(\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{x\})(\overline{x}(\vec{y}) \Pi_i Q_i \mid T \mid R) & & \\
\quad \equiv & (\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{x\})(\overline{x}(\vec{y}) \Pi_i Q_i \mid !x(\vec{y}).T' \mid R) & \\
\quad \mapsto & (\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{\vec{y}\})(\Pi_i Q_i \mid T' \mid !x(\vec{y}).T' \mid R) & \\
\quad \equiv & (\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{\vec{y}\})(\Pi_i Q_i \mid T' \mid R) & \\
\quad =_e & (\boldsymbol{\nu}\, \mathsf{fn}(A) \cup \{y_n\})(Q_n \mid R \mid T_1) & \text{(Prop.10 (4))} \\
\quad \in & \langle\!\langle \rho \rangle\!\rangle_{\xi, z}, & \text{(By (6))}
\end{array}$$

where $T_1 \stackrel{def}{=} (\boldsymbol{\nu}\, \vec{y} \backslash y_n)(T' \mid \Pi_{i \neq n}(\nu \mathsf{fn}(A))(Q_i \mid R))$, which is indeed in $\langle\!\langle \overline{\tau_n} \rangle\!\rangle_{\xi, y_n}$ by (5) and (7).

Case for $\forall$: Assume $\vdash_\phi P \triangleright A[x : \forall \mathrm{X}.\tau]$. We have two cases, either P is replication or linear input.

Subcase $\mathsf{md}(\tau) = \{\mathbf{!}\}$**.** Let $\mathsf{dom}(A) = \{x, \vec{y}\}$ where $A(y_i) = \tau_i$. By Proposition 9 (3) and $\mathrm{X} \notin \mathsf{ftv}(\tau')$, clearly:

$$\begin{array}{ll}
\forall \xi. \forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i}. (\boldsymbol{\nu}\, \vec{y})(P | \Pi_i Q_i) \in \langle\!\langle \tau_z \rangle\!\rangle_{\xi, z} & \\
\quad \Leftrightarrow \quad \forall \xi. \forall \mathcal{C}. \forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi \cdot \mathrm{X} \mapsto \mathcal{C}, y_i}. (\boldsymbol{\nu}\, \vec{y})(P | \Pi_i Q_i) \in \langle\!\langle \tau_z \rangle\!\rangle_{\xi \cdot \mathrm{X} \mapsto \mathcal{C}, z} & \\
\quad \Leftrightarrow \quad \forall \xi. \forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i}. \forall \mathcal{C}. (\boldsymbol{\nu}\, \vec{y})(P | \Pi_i Q_i) \in \langle\!\langle \tau_z \rangle\!\rangle_{\xi \cdot \mathrm{X} \mapsto \mathcal{C}, z} & \\
\quad \Leftrightarrow \quad \forall \xi. \forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i}. (\boldsymbol{\nu}\, \vec{y})(P | \Pi_i Q_i) \in \langle\!\langle \forall \mathrm{X}. \tau_z \rangle\!\rangle_{\xi, z}. &
\end{array}$$

By (IH) we are done.

Subcase $\mathsf{md}(\tau) = \downarrow$**.** Let $\mathsf{dom}(A) = \{x, y_1 ... y_n, z\}$ such that $A(z) = \tau_z^\uparrow$ and $A(y_i) = \tau_i$ for each i. Fix ξ and $Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi, y_i}$ $(1 \leq i \leq n)$. Starting from (IH), and noting $\langle\!\langle \tau \rangle\!\rangle_\xi$ is a candidate (by Proposition 9) as well as using the definition of $\perp$, we obtain:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall R \in \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi,x}.(\boldsymbol{\nu}\, x\vec{y})(P|\Pi_i Q_i|R) \in \langle\!\langle \tau_z \rangle\!\rangle_{\xi,z}} \\
\Leftrightarrow & \forall R \in \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi,x}.\forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Leftrightarrow & \forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}.\forall R \in \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi,x}.(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Leftrightarrow & \forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}.\forall R \in \bigcup_{\mathcal{C}} \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi\cdot \mathrm{X}\mapsto\mathcal{C},x}.(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Rightarrow & \forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}.(\boldsymbol{\nu}\, \vec{y}z)(P|\Pi_i Q_i|S) \in (\bigcup_{\mathcal{C}} \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi\cdot \mathrm{X}\mapsto\mathcal{C},x})^{\perp} \\
\Leftrightarrow & \forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}.\forall R \in (\bigcup_{\mathcal{C}} \langle\!\langle \overline{\tau} \rangle\!\rangle_{\xi\cdot \mathrm{X}\mapsto\mathcal{C},x})^{\perp\perp}.(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Leftrightarrow & \forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}\forall R \in \langle\!\langle \exists \mathrm{X}.\overline{\tau} \rangle\!\rangle_{\xi,x}.(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Leftrightarrow & \forall R \in \langle\!\langle \exists \mathrm{X}.\overline{\tau} \rangle\!\rangle_{\xi,x}\forall S \in \langle\!\langle \tau_z \rangle\!\rangle^{\perp}_{\xi,z}.(\boldsymbol{\nu}\, x\vec{y}z)(P|\Pi_i Q_i|R|S) \Downarrow_e \\
\Leftrightarrow & \forall R \in \langle\!\langle \exists \mathrm{X}.\overline{\tau} \rangle\!\rangle_{\xi,x}.(\boldsymbol{\nu}\, x\vec{y})(P|\Pi_i Q_i|R) \in \langle\!\langle \tau_z \rangle\!\rangle_{\xi,z}.
\end{array}
$$

Case for $\exists$: Assume $\vdash_\phi P \triangleright A[x : \exists \mathrm{X}.\tau]$ is derived from $\vdash_\phi P \triangleright A[x : \tau[\tau'/\mathrm{X}]]$. First, let $\mathsf{md}(\tau) \in \{\uparrow\}$, and $\mathsf{dom}(A) = \{x, \vec{y}\}$ where $A(y_i) = \tau_i$. We also assume x is a principal port. Starting from the (IH) and using Proposition 9 (2) and $\mathrm{X} \notin \mathsf{ftv}(\tau')$, we calculate:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall \xi.\forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi,y_i}.(\boldsymbol{\nu}\, \vec{y})(P|\Pi_i Q_i) \in \langle\!\langle \tau\{\tau'/\mathrm{X}\} \rangle\!\rangle_{\xi,y}} \\
\Leftrightarrow & \forall \xi.\forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi,y_i}.(\boldsymbol{\nu}\, \vec{y})(P|\Pi_i Q_i) \in \langle\!\langle \tau \rangle\!\rangle_{\xi\cdot \mathrm{X}\mapsto\langle\!\langle \tau' \rangle\!\rangle_\xi,y} \\
\Rightarrow & \forall \xi.\forall Q_i \in \langle\!\langle \overline{\tau_i} \rangle\!\rangle_{\xi,y_i}.(\boldsymbol{\nu}\, \vec{y})(P|\Pi_i Q_i) \in \langle\!\langle \exists \mathrm{X}.\tau \rangle\!\rangle_{\xi,y}
\end{array}
$$

as required. The case when $x : \tau$ is not principal is an exact dual to the corresponding $\forall$-case, hence is omitted.

Case $(\boldsymbol{\nu}\, \vec{w}) \prod_i P_i$: This term is inferred from $(\textsc{Par}_c)$. There are four cases for x principal:

(1)	$(\boldsymbol{\nu}\, \vec{w})(\overline{x}\langle\vec{y}\rangle \mid R)$	where x's mode is $\uparrow$
(2)	$(\boldsymbol{\nu}\, \vec{w})(\overline{y}\langle\vec{y}x\rangle \mid R)$	where y's mode is **?** and x's mode is $\uparrow$
(3)	$(\boldsymbol{\nu}\, \vec{w})(!x(\vec{y}).Q \mid R)$	where x's mode is **?** and x's mode is $\uparrow$
(4)	$(\boldsymbol{\nu}\, \vec{w})(b(\vec{y}).Q \mid R)$	where Q contains x and x's mode is $\uparrow$

Cases (1) and (2) are essentially as the same as the cases of the linear and replicated outputs above. Case (3) is also identical with the replication case above. For Case (4), we use the same proof method as that of [48, Lemma 3.8], using the acyclicity of name usage (Proposition 3) which is reflected on the acyclicity of causality chains. Then finally after applying $\mapsto$ following the strategy in [48, Lemma 3.8], it coincides either Case (1), Case (2) or $(\boldsymbol{\nu}\, \vec{w})(b(\vec{y}).Q' \mid R)$ where $b \notin \{\vec{w}\} \cup \mathsf{fn}(R)$ and Q' has a form of either Case (1) or (2). Then we repeat the same routine.

We have now exhausted all cases. □

5 Generic Transitions

This section discusses generic labelled transitions and their associated equivalences, another cornerstone of our theory. While our presentation focuses on the linear polymorphic π-calculus, the construction applies to other classes of behaviour. The duality principle strongly guides the construction. We begin with an informal sketch.

Setting up process semantics with a reduction relation is elegant but may not lead to tractable reasoning because the definition of the canonical congruence quantifies over all contexts: to prove P and Q are equal, we must conductively show that $C[P]$ and $C[Q]$ are observationally indistinguishable for all contexts $C[\cdot]$. While typing reduces the number of relevant (i.e. well-typed) contexts, this does not help much because there are still infinitely many to consider. This is where labelled transitions come to help. They can be considered a finite representation of those infinite sets of context, which makes reasoning much easier. But what are transitions for the second-order linear π-calculus?

The key difference from untyped and first-order calculi is that we now have two different forms of input transition (and, by duality, two forms of output).

1. The import of a name y on a channel x where the environment (partially) hides y's type by existential quantification. This restricts what we can do with y, but it does not constrain the names we may get via x: we must be prepared to input infinitely many names along x.
2. The *re-import* of a name x that we have exported earlier. Although the environment was restricted in what it was allowed to do with x by existential quantification, the process itself knows the concrete type of x. Since it is a re-import, we can only have exported a finite number of possible names, so there are only a finite (and in comparison with (1) small) number of names that can be inputted.

To put it crudely, in (1) we must be ready to receive many names, but we know very little about them, while in (2), we can only get a few names, but we know a lot about them. Observe how the duality in type structures plays a fundamental role.

For reasoning, receiving fewer names is better because it means fewer transitions, which in turn means fewer cases to check. On the other hand, knowing a name's full type is advantageous, because we can convert free into bound outputs plus forwarding. Bound outputs are easier to reason about because we can always assume that all output names are distinct and not occurring in their recipients. The transition system to be presented tries to take advantage of both phenomena as much as possible, but there's a trade-off with genericity: if a process hides a name x by existential quantification, it removes type-information which prevents x's recipients to replace free outputs $\overline{y}\langle x\rangle$ by bound ones $\overline{y}(x')[x' \to x]$ where $[x' \to x]$ is copy-cat agent [11, 21, 24, 48, 49], which links two names between x' and x, for example $[x' \to x] = x'.\overline{x}$. There is simply not enough type information to know what type $[x' \to x]$ should forward. The idea of the labelled transition system below is to use bound outputs as much as possible while having as few transitions as possible. To achieve this, we 'instrument' the typing system with additional information about whether a process uses a name as described by (1) or by (2). The key problem is to track the "change of types" as an existentially abstracted name flows from its producer to consumers and back. We do this by having two forms of type variables $\mathrm{X}^{\exists}_{\tau}$ and $\mathrm{X}^{\forall}$. If x has type $\mathrm{X}^{\forall}$, then x was received and its sender has hidden the true type by existential abstraction. Then we cannot use copycats to forward this name. We must freely output x instead. If x has type $\mathrm{X}^{\exists}_{\tau}$ then we are its "producer". We know that x's true type is τ, but to the outside we export it typed by a type variable, preventing all recipients from using x as a subject in communication. But it can be emitted in bound form. By way of an environment we keep track of which names we have already exported under existential quantification, so our transitions are types. The type information is used to decide whether a name is output freely or not. It also restricts the set of possible input subjects when reimporting existentially abstracted names. We are now continuing by making this sketch precise.

5.1 Types for Generic Transitions

The transition uses an extended typing where type variables in action types are annotated by quantification symbols (as $\mathrm{X}^{\forall}$ and $\mathrm{X}^{\exists}$, called *universal type variable* and *existential type variable*, respectively). The original free type variables and $\forall$-quantified variables are naturally $\forall$-annotated, while $\exists$-quantified variables are $\exists$-annotated. Formally, types are now generated by the following grammar.

$$\tau ::= \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \mid \updownarrow \qquad \tau_{\mathtt{I}} ::= \mathrm{X}^{\forall} \mid \mathrm{X}^{\exists}_{\tau_{\mathtt{I}}} \mid \forall \vec{\mathrm{X}}.\tau_{\mathtt{I}} \qquad \tau_{\mathtt{O}} ::= \mathrm{X}^{\forall} \mid \mathrm{X}^{\exists}_{\tau_{\mathtt{O}}} \mid \exists \vec{\mathrm{X}}.\tau_{\mathtt{O}}$$

We assume $\forall \vec{\mathrm{X}}.\tau$ only binds free $\forall$-annotated variable $\mathrm{X}^{\forall}_{i}$ in τ. Dually for exists. In $\mathrm{X}^{\exists}_{\tau}$, τ is called its *concrete type annotation.* Concrete type annotations are only for free existential type variables, and are *not* part of the type structure proper, in the sense that (among others) they are always neglected in behavioural equivalences. They are however useful for having a simpler definition of generic transition (the way to induce the same transition relation without concrete type annotations is outlined later). On the other hand, $\exists$ and $\forall$ annotations are the intrinsic part of the type structure, and plays a fundamental role in characterising generic behaviour. In examples, we shall omit concrete type annotations unless they are absolutely necessary.

We need to refine the definition of dualisation as follows.

$$\overline{\mathrm{X}^{\forall}} = \overline{\mathrm{X}}^{\forall} \qquad \overline{\mathrm{X}^{\exists}_{\tau}} = \overline{\mathrm{X}}^{\exists}_{\overline{\tau}} \qquad \overline{\forall \vec{\mathrm{X}}.\tau} = \exists \vec{\mathrm{X}}.\overline{\tau\{\vec{\mathrm{X}}^{\exists}/\vec{\mathrm{X}}^{\forall}\}} \qquad \overline{\exists \vec{\mathrm{X}}.\tau} = \forall \vec{\mathrm{X}}.\overline{\tau\{\vec{\mathrm{X}}^{\forall}/\vec{\mathrm{X}}^{\exists}\}}$$

For example $\overline{\forall \mathrm{X}.(\mathrm{X}^{\forall}\mathrm{Y}^{\forall}\mathrm{Z}^{\exists}_{\tau}(\overline{\mathrm{X}}^{\forall})\downarrow)^{?}} = \exists \mathrm{X}.(\overline{\mathrm{X}}^{\exists}\overline{\mathrm{Y}}^{\forall}\overline{\mathrm{Z}}^{\exists}_{\overline{\tau}}(\mathrm{X}^{\exists})\uparrow)^{!}$. The definitions of $\odot$ and $\asymp$ are unchanged, using the new dualisation. We also extend $\mathsf{ftv}(\cdot)$ and add a function $\mathsf{ftv}^{\exists}(\cdot)$ that returns only existentially-annotated type variables: $\mathsf{ftv}(\mathrm{X}^{\forall}) = \{\mathrm{X}^{\forall}, \overline{\mathrm{X}^{\forall}}\}$, $\mathsf{ftv}(\mathrm{X}^{\exists}_{\tau}) = \{\mathrm{X}^{\exists}, \overline{\mathrm{X}^{\exists}}\} \cup \mathsf{ftv}(\tau)$, $\mathsf{ftv}^{\exists}(\mathrm{X}^{\forall}) = \emptyset$, $\mathsf{ftv}^{\exists}(\mathrm{X}^{\exists}_{\tau}) = \{\mathrm{X}^{\exists}, \overline{\mathrm{X}^{\exists}}\} \cup \mathsf{ftv}^{\exists}(\tau)$. The non-base cases and the extension to general action types are obviously given. The typing rules is found in Figure 2. Free $\exists$-type variables are introduced by

$$\frac{(\text{In}^{\downarrow})\quad (\mathrm{x}_i^{\forall} \notin \mathsf{ftv}(A,B)) \qquad \vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}\ ,\uparrow A^{-x}, ?B^{-x}}{\vdash_{\mathtt{I}} x(\vec{y}).P \triangleright (x:\forall\vec{\mathrm{X}}.(\vec{\tau})^{\downarrow} \to A), B}$$

$$\frac{(\text{In}^{!})\quad (\mathrm{x}_i^{\forall} \notin \mathsf{ftv}(A)) \qquad \vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}, ?A^{-x}}{\vdash_{\mathtt{I}} !x(\vec{y}).P \triangleright x:\forall\vec{\mathrm{X}}.(\vec{\tau})^{!} \to A}$$

$$\frac{(\text{Out})\quad (\mathsf{ftv}^{\exists}(\vec{\sigma}) = \emptyset) \qquad \sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}^{\exists}_{\vec{\gamma}}\}}{\vdash_{\mathtt{O}} \overline{x}\langle\vec{y}\rangle \triangleright x:\exists\vec{\mathrm{X}}.(\vec{\tau})^{p_{\mathtt{O}}};\ \vec{y}:\overline{\vec{\sigma}}}$$

$$\frac{(\exists\text{-Var})\quad \mathrm{X}^{\exists} \notin \mathsf{ftv}(\tau) \qquad \vdash_{\phi} P \triangleright A\{\tau/\mathrm{X}^{\exists}_{\tau}\}}{\vdash_{\phi} P \triangleright A} \qquad \frac{(\forall\text{-Var})\quad \mathrm{X}^{\forall} \notin \mathsf{ftv}(\tau) \qquad \vdash_{\phi} P \triangleright A}{\vdash_{\phi} P \triangleright A\{\tau/\mathrm{X}^{\forall}\}}$$

Fig. 2. Extended polymorphic sequential typing for generic transitions. The remaining rules (Zero), (Par), (Res) and (Weak) have been omitted as they are unchanged from Figure 1.

($\exists$-Var); ($\forall$-Var) is its dual. From now on, expressions of the form $\vdash_{\phi} P \triangleright A$ will always refer to this new typing systems, except where stated otherwise.
As an example of typing, we have, in the extended typing:

$$\vdash_{\mathtt{O}} \mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle \triangleright y:\mathrm{X}^{\exists}, z:(\overline{\mathrm{X}^{\exists}})^{\uparrow}, e:(\overline{\mathbb{B}})^{\downarrow},$$

which abstracts away the type which is both for the resource at y and for the value of the input via z.

Proposition 11. *If* $\vdash_{\phi} P \triangleright A$ *and* $P \longrightarrow Q$, *then* $\vdash_{\phi} Q \triangleright A$.

Proof. Straightforward by the substitution lemma. See Appendix D.1. □

5.2 Generic Transitions

We first give the set of action labels $(l, l', \ldots)$ by the following grammar.

$$l \ ::= \ x\langle(\nu\vec{y})\vec{z}\rangle \mid \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \mid \tau$$

In the first two labels, the vector $\vec{y}$ is made up from pairwise distinct names and forms a not necessarily consecutive subsequence of $\vec{z}$. The *subject* x must not occur in $\vec{y}$. The names in $\vec{z}$ are *objects*, while those in $\vec{y}$ occur *bound*. Names not being bound occur *free*. $x(\vec{y})$ and $x\langle\vec{y}\rangle$ stand for $x\langle(\nu\vec{y})\vec{y}\rangle$ and $x\langle(\nu\varepsilon)\vec{y}\rangle$, respectively, similarly for output actions.

As in various other typed transition relations for the π-calculus, the essence of generic transition lies in the reduction of possible transitions to the bare minimum ones for faithfully representing typed behaviour (or, more simply put, interactions between typed processes). For this purpose the following predicates decide possible shape of action labels conforming to a given action type. Among others, the predicate dictates that a free output (resp. input) corresponds to a name typed by a universal type variable (resp. existential type variables), following the idea sketched at the outset of this section. Formally the *typed action predicate* $A \vdash l$ is given by the following three rules.

– $A \vdash \tau$ always.
– $A \vdash x\langle(\nu\vec{z})\vec{w}\rangle$ when $\{\vec{z}\} \cap \mathsf{fn}(A) = \emptyset$ and $A(x) = \forall\vec{\mathrm{X}}.(\vec{\tau})^{p_{\mathtt{I}}}$ s.t. $w_i \notin \{\vec{z}\}$ iff $A(w_i) = \overline{\tau_i}$ where τ_i is an existential type variable.
– $A \vdash \overline{x}\langle(\nu\vec{z})\vec{w}\rangle$ when $\{\vec{z}\} \cap \mathsf{fn}(A) = \emptyset$ and $A(x) = \exists\vec{\mathrm{X}}.(\vec{\tau})^{p_{\mathtt{O}}}$ s.t. $w_i \notin \{\vec{z}\}$ iff $A(w_i) = \overline{\tau_i}$ where τ_i is a universal type variable.

Let A be an action type and l a label. We say A *allows* l (cf. [11, 48]) when:

– $\mathsf{bn}(l) \cap \mathsf{fn}(A) = \emptyset$; and
– at least one of the following is true.
  - l is an output with subject x and $\mathsf{md}(A(x)) = \mathbf{?}$;
  - $l = \tau$; or
  - l is a visible action such that $\mathsf{sbj}(l) \notin \mathsf{fn}(A)$.

This definition means that we do not allow a linear input action and an output action when there is a complementary channel in the process. For example, if a process has $x : \updownarrow$ (resp. $x : (\vec{\tau})^{!}$) in its action type, then both input and output actions (resp. output) at x should be excluded since such actions can never be observed in a typed context [48].

Preview of the Generic Transition System. We are now ready to introduce the transition rules. To simplify things we start by giving just dealing with the monadic case at first. We start from the standard bound input rule.

$$(\text{BIN}^{\downarrow}) \quad \vdash_{\mathtt{I}} x(y).P \triangleright A \ \xrightarrow{x(y)}\ \vdash_{\mathtt{O}} P \triangleright A/x, y : \tau \qquad (A \vdash x(y))$$

This may introduce (output-moded) $\forall$-type variables, which are used as follows.

$$(\text{FOUT}^{\uparrow}) \quad \vdash_{\mathtt{O}} \overline{x}\langle y\rangle \triangleright A \ \xrightarrow{\overline{x}\langle y\rangle}\ \vdash_{\mathtt{I}} 0 \triangleright A/x \qquad (A \vdash \overline{x}\langle y\rangle)$$

We can now infer, using the replicated variant of the input rule,

$$\vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle \triangleright x : \mathbb{I} \ \xrightarrow{x(yz)}\xrightarrow{\overline{z}\langle y\rangle}\ \vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle \triangleright x : \mathbb{I} \mid 0.$$

Next we consider the dual situation of the above, starting from bound output.

$$(\text{BOUT}^{\uparrow}) \quad \vdash_{\mathtt{O}} \overline{x}\langle w\rangle \triangleright A \ \xrightarrow{\overline{x}(z)}\ \vdash_{\mathtt{I}} [z \to w]^{\sigma} \triangleright \vec{z} : \sigma \to w : \overline{\sigma}$$

with $A \vdash \overline{x}(z^{\sigma})$. Here $[z \to w]^{\tau}$ is copy-cat agent [11, 21, 24, 48, 49], which links two names between z and w. For example, $[z \to w]^{\tau} = !z(x).\overline{w}(y)y.\overline{x}$ with $\tau = (()^{\uparrow})^{!}$ (the $[z \to w]^{\tau}$ agent is formally defined below). This rule is best seen in view of the following semantic equality between free and bound name passing.

$$\overline{x}\langle y\rangle \cong_{\forall\exists} \overline{x}(z)[z \to y]^{\tau}.$$

where $\cong_{\forall\exists}$ is the standard contextual equality defined in §.5.3 (cf. Lemma 1 for the formal statement). Thus, by replacing a free output with a more abstract bound output combined with copy-cats, we precisely obtain the effect of the above transition.

($\text{BOut}^{\uparrow}$) may introduce (input-moded) $\exists$-type variables, which are used by:

$$(\text{FIn}^{\downarrow})\ \vdash_{\mathtt{I}} x(y).P \triangleright A \xrightarrow{x\langle z\rangle} \vdash_{\mathtt{O}} P\{z/y\} \triangleright A/x \odot z : \overline{\mathrm{X}} \quad (A \odot z : \overline{\mathrm{X}} \vdash x\langle z\rangle)$$

In the side condition above, we pre-compose $z:\overline{\mathrm{X}}$ (assuming $A \asymp z:\overline{\mathrm{X}}$ by the side condition), which are types for opaque resources, and which are to be composed later (for illustration of this point see the example below). This rule says that an input may receive channels for opaque resources which have been exported and which, therefore, are free.

The pre-composition of resource types in the above rule deserves illustration. The following is the example of inference using the replicated variant of ($\text{BOut}^{\uparrow}$) and ($\text{FIn}^{\downarrow}$) (note the following term corresponds to the examples in Section 2).

$$\begin{aligned}
&\vdash_{\mathtt{O}}\ \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(w).R) \triangleright x:\overline{\mathbb{I}}, e:(\mathbb{B})^{\uparrow}\\
&\qquad\qquad \xrightarrow{\overline{x}(yz)} \vdash_{\mathtt{O}}\ \mathsf{t}\langle y\rangle | z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle \triangleright y:\overline{\mathrm{X}}^{\exists}, z:(\overline{\mathrm{X}}^{\exists})^{\uparrow}, e:(\overline{\mathbb{B}})^{\downarrow}\\
&\qquad\qquad \xrightarrow{z\langle y\rangle} \vdash_{\mathtt{O}}\ \mathsf{t}\langle y\rangle | \overline{e}(c)\mathsf{not}\langle cy\rangle \triangleright y:\overline{\mathrm{X}}^{\exists}, e:(\mathbb{B})^{\uparrow}.
\end{aligned}$$

In the second action, the base case of the derivation is:

$$\vdash_{\mathtt{I}}\ z(w).\overline{e}(c)\mathsf{not}\langle cw\rangle \triangleright z:(\overline{\mathrm{X}}^{\exists})^{\uparrow}, e:(\overline{\mathbb{B}})^{\downarrow} \xrightarrow{z\langle y\rangle} \vdash_{\mathtt{O}}\ \overline{e}(c)\mathsf{not}\langle cy\rangle \triangleright y:\overline{\mathrm{X}}^{\exists}, e:(\mathbb{B})^{\uparrow}$$

Note that the term does (and can) *not* own the input type at y at the time of this derivation. To derive the action, we need to pre-compose $y:\overline{\mathrm{X}}^{\exists}$ to the base, so that we have the following action predicate valid:

$$z:(\overline{\mathrm{X}}^{\exists})^{\uparrow}, e:(\overline{\mathbb{B}})^{\downarrow}, y:\overline{\mathrm{X}}^{\exists}\ \vdash\ z\langle y\rangle$$

which allows us to have the transition. In the final configuration, we find $\mathsf{t}\langle y\rangle$ of type $y:\mathrm{X}^{\exists}$ gets composed, just as predicted by the pre-composition. This illustrates the need of precomposition of resource types in ($\text{FIn}^{\downarrow}$). Before the formal definition, we define the copy-cat agent, which is also used in Section 7.

It is well-known [48] that one can often replace free by bound outputs with forwarding: $\overline{x}\langle y\rangle$ is then observationally indistinguishable from $\overline{x}(z)[z \to y]$. In the present calculus, we can always do this as long as the carried name is not typed by a universally annotated type variable. The reason is that in order to define appropriate forwarders, we need to know what types of data a name is intended to carry. If the only information about a name is that it is typed by $\mathrm{X}^{\forall}$ then we're not sure how to forward. In this case, it must remain a free output. As bound outputs are mathematically easier to deal with than free ones – because we can always assume that all the names we receive will be fresh from the point of view of the receiving process, we will often restrict our attention to processes that have as few free outputs as possible. We now provide tools that make this sentiment formal. Let σ be an input-moded type that is not a type variable. Then $[x \to y]^{\sigma}$ is defined inductively:

$$[x \to y]^{\forall\vec{\mathrm{X}}.(\vec{\tau})^{\downarrow}} \stackrel{def}{=} x(\vec{v}).\overline{y}\langle\!\langle\vec{v}\rangle\!\rangle^{\overline{\vec{\tau}}} \qquad [x \to y]^{\forall\vec{\mathrm{X}}.(\vec{\tau})^{!}} \stackrel{def}{=} !x(\vec{v}).\overline{y}\langle\!\langle\vec{v}\rangle\!\rangle^{\overline{\vec{\tau}}}$$

where w.l.o.g. the $\tau_0, ..., \tau_{i-1}$ are all universally quantified type variables, while none of $\tau_i, ..., \tau_{n-1}$ is. Here

$$\overline{y}\langle\!\langle \vec{v} \rangle\!\rangle^{\vec{\tau}} \stackrel{def}{=} (\nu w_i...w_{n-1})(\overline{y}\langle v_0...v_{i-1}w_i...w_{n-1}\rangle \mid \Pi_{j=i}^{n-1}[w_j \to v_j]^{\mathsf{con}(\tau_j)}).$$

If $\vdash_\phi P \triangleright A$, then $\langle\!\langle P \rangle\!\rangle^{\phi,A}$ is obtained from P by replacing every output $\overline{x}\langle\vec{y}\rangle$ in P with $\overline{x}\langle\!\langle \vec{y} \rangle\!\rangle^{\vec{\tau}}$. We omit further details of the definition. P is *maximally copycatted* if $P = \langle\!\langle P \rangle\!\rangle^{\phi,A}$.

Lemma 10. *1. If* $\vdash_0 \overline{x}\langle\vec{y}\rangle \triangleright x : \exists\vec{\mathrm{X}}.(\vec{\tau})^{p_0}, E$ *then* $\overline{x}\langle\vec{y}\rangle \cong_{\forall\exists} \overline{x}\langle\!\langle \vec{y} \rangle\!\rangle^{\vec{\tau}}$.
2. If $\vdash_\phi P \triangleright A$, *then* $\vdash_\phi \langle\!\langle P \rangle\!\rangle^{\phi,A} \triangleright A$, $\vdash_\phi P \cong_{\forall\exists} \langle\!\langle P \rangle\!\rangle^{\phi,A} \triangleright A$ *and* $\langle\!\langle P \rangle\!\rangle^{\phi,A}$ *is maximally copycatted.*

Proof. Straightforward. □

The *generic transition relation* $\xrightarrow{l}$ is formally defined in Figure 3. It is built on top of *pretransitions* $\overset{l}{\rightsquigarrow}$, also defined there. Since a type may carry both type variable(s) and concrete type(s), these rules combine free actions and bound actions (so (IN$^\downarrow$), resp. (OUT$^\uparrow$), combines (BIN$^\downarrow$) and (FIN$^\downarrow$), resp. (BOUT$^\uparrow$) and (FOUT$^\uparrow$)). In (OUT), we use the following function, $\mathsf{con}(\cdot)$, when the object name is an existential variable: $\mathsf{con}(\mathrm{X}^\exists_\tau) = \tau$, and $\mathsf{con}(\tau) = \tau$ with $\tau \neq \mathrm{X}^\exists$. (FULL) is necessary to preserve compositionality of free linear name passing. A simpler, but less compositional alternative transition system can be found in Appendix E.

For the generated transition relation, we can check the following result. Below and henceforth the typability is taken under the extended typing.

Proposition 12. (subject transition) *Let* $\vdash_\phi P \triangleright A$. *If* $\vdash_\phi P \triangleright A \overset{l}{\rightsquigarrow} \vdash_\psi Q \triangleright B$, *then* $\vdash_\psi Q \triangleright B$. *Similarly if* $\vdash_\phi P \triangleright A \xrightarrow{l} \vdash_\psi Q \triangleright B$, *then* $\vdash_\psi Q \triangleright B$.

Proof. See Appendix D.2. □

Remark 2. (transitions without concrete type annotations) Given a typed process, we can turn all its occurring non-generic free outputs into bound outputs augmented with copycats. Once we do this transformation, the syntactic transformation in (OUT) becomes unnecessary. Since (OUT) is the sole place where we need concrete type annotations associated with existentially annotated type variables, this preprocessing allows us to dispense with these annotations. The presented transition however has the merit in that it can induce the generic transition relation from arbitrary typed terms, not only preprocessed ones.

5.3 Generic Bisimilarity

We are now going to define a typed bisimulation. Since concrete type annotations on free existential variables are not part of the type structure (cf. §5.1). we should consider typed relations neglecting the annotations from the types of

$$(\textsc{In}^{\downarrow})\quad \frac{A \asymp \vec{w} : \overline{\vec{\tau}} \qquad A/x \asymp \vec{w} : \vec{\tau} \qquad A \odot \vec{w} : \overline{\vec{\tau}} \vdash x^{\downarrow}\langle(\nu\vec{z})\vec{w}^{\vec{\tau}}\rangle}{\vdash_{\mathtt{I}} x(\vec{y}).P \rhd A \xrightarrow{x\langle(\nu\vec{z})\vec{w}\rangle} \vdash_{\mathtt{O}} P\{\vec{w}/\vec{y}\} \rhd A/x \odot \vec{w} : \vec{\tau}}$$

$$(\textsc{In}^{!})\quad \frac{A \asymp \vec{w} : \overline{\vec{\tau}} \qquad A \asymp \vec{w} : \vec{\tau} \qquad A \odot \vec{w} : \overline{\vec{\tau}} \vdash x^{!}\langle(\nu\vec{z})\vec{w}^{\vec{\tau}}\rangle}{\vdash_{\mathtt{I}} \;!x(\vec{y}).P \rhd A \xrightarrow{x\langle(\nu\vec{z})\vec{w}\rangle} \vdash_{\mathtt{O}} \;!x(\vec{y}).P|P\{\vec{w}/\vec{y}\} \rhd A \odot \vec{w} : \vec{\tau}}$$

$$(\textsc{Out})\quad \frac{\sigma_i = \mathsf{con}(\rho_i) \qquad A \vdash \overline{x}\langle(\nu\vec{z}^{\vec{\rho}})\vec{w}^{\vec{\tau}}\rangle}{\vdash_{\mathtt{O}} \overline{x}\langle\vec{w}\rangle \rhd A \xrightarrow{\overline{x}\langle(\nu\vec{z})\vec{w}\{\vec{z}/\vec{y}\}\rangle} \vdash_{\mathtt{I}} \Pi_i[z_i \to y_i]^{\sigma_i} \rhd \vec{z} : \vec{\sigma} \to \vec{y} : \overline{\vec{\sigma}}}$$

$$(\textsc{Com})\quad \frac{\vdash_{\phi} P_1 \rhd A_1\{\vec{\tau}/\vec{\mathrm{X}}^{\forall}\} \overset{l}{\leadsto} \vdash_{\psi} P_1' \rhd A_1' \qquad \vdash_{\overline{\phi}} P_2 \rhd A_2\{\vec{\tau}/\vec{\mathrm{X}}^{\forall}\} \overset{\overline{l}}{\leadsto} \vdash_{\overline{\psi}} P_2' \rhd A_2' \qquad A_1 \asymp A_2}{\vdash_{\mathtt{O}} P_1|P_2 \rhd A_1 \odot A_2 \overset{\tau}{\leadsto} \vdash_{\mathtt{O}} (\boldsymbol{\nu}\,\mathsf{bn}(l))(P_1'|P_2') \rhd A_1 \odot A_2}$$

$$(\textsc{Par})\quad \frac{\vdash_{\phi} P_1 \rhd A_1 \overset{l}{\leadsto} \vdash_{\psi} P_1' \rhd A_1' \qquad \vdash_{\mathtt{I}} P_2 \rhd A_2 \qquad A_1 \asymp A_2 \qquad l \text{ allowed by } A_2}{\vdash_{\phi} P_1|P_2 \rhd A_1 \odot A_2 \overset{l}{\leadsto} \vdash_{\psi} P_1'|P_2 \rhd A_1' \odot A_2}$$

$$(\textsc{Res})\quad \frac{\vdash_{\phi} P \rhd A \overset{l}{\leadsto} \vdash_{\psi} Q \rhd B \qquad x \notin \mathsf{fn}(l)}{\vdash_{\phi} (\boldsymbol{\nu}\,x)P \rhd A/x \overset{l}{\leadsto} \vdash_{\psi} (\boldsymbol{\nu}\,x)Q \rhd B/x}$$

$$(\textsc{Full})\quad \frac{A \vdash l \qquad \vdash_{\phi} P \rhd A \overset{l}{\leadsto} \vdash_{\psi} Q \rhd B}{\vdash_{\phi} P \rhd A \xrightarrow{l} \vdash_{\psi} Q \rhd B}$$

Fig. 3. Generic transition rules.

processes. We write $|A|$ for the result of taking off concrete type annotations from A. $\mathcal{A}, \mathcal{B}, \ldots$ range over action types without concrete type annotations. *For avoiding the clattering of notation, we shall henceforth often confuse action types with concrete type annotations and those which are the result of taking them off, except for formal definition which follows.*

A *typed relation* $\mathbf{R}$ is a collection of ($\mathbf{R}_{\phi,\mathcal{A}}$ which is a binary relations on processes such that $P_1 \mathbf{R}_{\phi,\mathcal{A}} P_2$ implies $\vdash_{\phi} P_i \rhd A_i$ with $|A_i| = \mathcal{A}_i$ for $i = 1, 2$. We often write $\vdash_{\phi} P \;\mathbf{R}\; Q \rhd \mathcal{A}$ instead of $P\;\mathbf{R}_{\phi,\mathcal{A}}\;Q$. $\mathbf{R}$ is a *typed equivalence* if it is reflexive (on typed processes), symmetric and transitive. $\mathbf{R}$ is a *typed congruence* if it is a typed equivalence and closed under the typing rules. This means for example that $\vdash_{\mathtt{O}} P\mathbf{R}Q \rhd \vec{y} : \vec{\tau}, \uparrow A^{\text{-}x}, ?B^{\text{-}x}$ implies $\vdash_{\mathtt{I}} x(\vec{v}).P \;\;\mathbf{R}\;\; x(\vec{v}).Q \rhd x : \forall\vec{\mathrm{X}}.(\vec{\tau})^{\downarrow}, \uparrow A^{\text{-}x}, ?B^{\text{-}x}$, provided that $\{\vec{\mathrm{X}}\} \cap \mathsf{ftv}(A, B) = \emptyset$.

A relation $\mathbf{R}$ between typed processes of the same action type and mode in the extended typing is a *generic weak bisimulation* (alternatively *polymorphic weak bisimulation* (cf. [37]) or simply a *weak bisimulation*) if whenever $\vdash_{\phi} P_1 \mathbf{R} P_2 \rhd \mathcal{A}$

with $|A_1| = |A_2| = \mathcal{A}$, the following and the symmetric case hold: $\vdash_\phi P_1 \triangleright A_1 \xrightarrow{l} \vdash_\phi P_1' \triangleright A_1'$ implies $\vdash_\phi P_2 \triangleright A_2 \stackrel{\hat{l}}{\Longrightarrow} \vdash_\phi P_1' \triangleright A_2'$ such that $\vdash_{\phi'} P_1' \mathbf{R} P_2' \triangleright \mathcal{A}'$ again with $\mathcal{A}' = |A_1| = |A_2|$. The largest weak bisimulation exists, which we call *weak bisimilarity* and denote by $\approx_{\forall\exists}$. We write $\vdash_\phi P \approx_{\forall\exists} Q \triangleright \mathcal{A}$ or $P \approx_{\forall\exists}^{\mathcal{A},\phi} Q$ if P and Q are bisimilar under $\mathcal{A}, \phi$. From now on we confuse $\mathcal{A}$ and its concrete instances, simply writing $\vdash_\phi P \approx_{\forall\exists} Q \triangleright A$ and $P \approx_{\forall\exists}^{A,\phi} Q$

Proposition 13. $\approx_{\forall\exists}$ *is a typed congruence.*

Proof. For example, the closure under parallel composition is proved by defining $\mathbf{R}$ by the following rule: if $\vdash_\phi P_1 \approx_{\forall\exists} P_2 \triangleright A$ and $\vdash_\psi R \triangleright B$ such that $\vdash_{\phi\odot\psi} (\boldsymbol{\nu}\,\vec{y})(P_i|R) \triangleright (A \odot B)/\vec{z}$ is well-typed, then $(\boldsymbol{\nu}\,\vec{y})(P_1|R)\mathbf{R}(\boldsymbol{\nu}\,\vec{y})(P_2|R)$. Then we can easily check $\mathbf{R}$ is a bisimulation. The reasoning is standard and is omitted. $\square$

Lemma 11. $\equiv$ *and* $\mapsto\!\!\!\rightarrow$ *are both weak bisimulations.*

Proof. We show that typable processes related by $\equiv$ have the same transitions up to $\equiv$ by induction on the derivation of $\equiv$ with a nested induction on the derivation of transitions. To see that $\twoheadrightarrow$ is a weak bisimulation, we induce on the generation of $\twoheadrightarrow$ with a nested induction on the structure of the used contexts. Congruency of $\approx_{\forall\exists}$ (Lemma 13) is used, too. $\square$

The following lemma says that reductions and τ-transitions coincide up to $\equiv$.

Lemma 12. *Let* $\vdash_\phi P \triangleright A$*, then* $P \longrightarrow Q$ *iff* $\vdash_\phi P \triangleright A \xrightarrow{\tau} \vdash_\phi Q' \triangleright A$ *for some* $Q' \equiv Q$*.*

Proof. By straightforward inductions on the derivation of the transitions and reductions with nested inductions on the typing judgement. $\square$

Proposition 14. (soundness) $\vdash_\phi P \approx_{\forall\exists} Q \triangleright A$ *implies* $\vdash_\phi P \cong_{\forall\exists} Q \triangleright A$*.*

Proof. As clearly $\overline{x}\langle\vec{y}\rangle \not\approx_{\forall\exists} 0$, $\approx_{\forall\exists}$ is consistent. By Proposition 13 it is a congruence. For reduction-closure, let $\vdash_\phi P \triangleright A$, $P \approx_{\forall\exists} Q$ and $P \longrightarrow P'$. By Lemma 12 then $\vdash_\phi P \triangleright A \xrightarrow{\tau} \vdash_\phi P'' \triangleright A$ for some $P' \equiv P''$. Then there's a transition sequence $Q \xrightarrow{\tau} \cdots \xrightarrow{\tau} Q' \approx_{\forall\exists} P'' \equiv P'$. The result follows because $\equiv, \approx_{\forall\exists} \circ \approx_{\forall\exists} \subseteq \approx_{\forall\exists}$ (Lemma 11).

It reminds to show preservation of barbs. Let $\vdash_\phi P \approx_{\forall\exists} Q \triangleright x : \mathbb{B}$ and $P \mapsto\!\!\!\rightarrow \mathsf{t}\langle x\rangle$. Then $Q \approx_{\forall\exists} P \approx_{\forall\exists} \mathsf{t}\langle x\rangle$ (Lemma 11). Since $\mapsto$ is SN (Theorem 2), we can find Q' such that $\mathsf{t}\langle x\rangle \approx_{\forall\exists} Q \mapsto\!\!\!\rightarrow Q' \not\mapsto$. Applying Lemma 6 ensures that $Q' \equiv \mathsf{t}\langle x\rangle$ or $Q' \equiv \mathsf{f}\langle x\rangle$. But a quick look at the transitions shows that we cannot have $\mathsf{f}\langle x\rangle \approx_{\forall\exists} \mathsf{t}\langle x\rangle$, so we cannot have $Q' \equiv \mathsf{f}\langle x\rangle$. $\square$

We now define the contextual congruence. Let $\vdash_{\mathtt{I}} P \triangleright x : \mathbb{B}$. We write $P \Downarrow_{\mathsf{f}\langle x\rangle}$ iff $P \mapsto\!\!\!\rightarrow \mathsf{f}\langle x\rangle$, and $P \Downarrow_{\mathsf{t}\langle x\rangle}$ iff $P \mapsto\!\!\!\rightarrow \mathsf{t}\langle x\rangle$. Now $\cong_{\forall\exists}$ is the largest consistent, reduction-closed, typed congruence that preserves $\Downarrow_{\mathsf{t}\langle x\rangle}$ and $\Downarrow_{\mathsf{f}\langle x\rangle}$.

5.4 Legal Traces

The generic transition sequences of polymorphic affine processes enjoy the ordered structure which extends that of their first-order subset [11]. This leads to the characterisation of behaviour of generic processes by a function of some kind.

Let $s = l_0 ... l_{n-1}$. We write $\vdash_\phi P \triangleright A \stackrel{s}{\Longrightarrow} \vdash_\psi Q \triangleright B$ if

$$\vdash_\phi P \triangleright A \xrightarrow{\tau}{}^* \xrightarrow{l_0} \xrightarrow{\tau}{}^* \cdots \xrightarrow{\tau}{}^* \xrightarrow{l_{n-1}} \xrightarrow{\tau}{}^* \vdash_\psi Q \triangleright B$$

We first define a subset of typed transition sequences, which we call *legal*. We recall some ideas from [11], which originally comes from game semantics. Below we assume $l, ..$ do not include τ-action. We write $l \curvearrowright_{\mathsf{s}} l'$ (resp. $l \curvearrowright_{\mathsf{o}} l'$) if the binder of l binds the subject (resp. a free object) of l' (the object binding is new in generic transition, when compared with the definition in [11]). Typing guarantees that we cannot have $l \curvearrowright_{\mathsf{s}} l'$ and $l \curvearrowright_{\mathsf{o}} l'$ at the same time. We then set $\curvearrowright = \curvearrowright_{\mathsf{s}} \cup \curvearrowright_{\mathsf{o}}$. These three relations are extended to strings in the the obvious way, e.g. $s \curvearrowright_{\mathsf{s}} l$ iff $s = s_1 l' s_2$ and $l' \curvearrowright_{\mathsf{s}} l$. These binding relations essentially preserve prefixing information available, that would otherwise be lost when we abstract from $P \xrightarrow{l} \cdots \xrightarrow{l'} P'$ to sequences $l, ..., l'$. The *O-view* of a sequence $\ulcorner l_1 ... l_n \urcorner^{\mathtt{O}}$ is defined as follows, with $s, t, \ldots$ ranging over sequences of labels.

$$\begin{array}{lll}
\ulcorner \epsilon \urcorner^{\mathtt{O}} & = \emptyset & \\
\ulcorner s \cdot l_n \urcorner^{\mathtt{O}} & = \{n\} \cup \ulcorner s \urcorner^{\mathtt{O}} & l_n \text{ output} \\
\ulcorner s \cdot l_n \urcorner^{\mathtt{O}} & = \{n\} & l_n \text{ input}, \forall i. l_i \not\curvearrowright_{\mathsf{s}} l_n \\
\ulcorner s_1 \cdot l_i \cdot s_2 \cdot l_n \urcorner^{\mathtt{O}} & = \{i, n\} \cup \ulcorner s_1 \urcorner^{\mathtt{O}} & l_n \text{ input}, l_i \curvearrowright_{\mathsf{s}} l_n
\end{array}$$

The *Input view*, denoted $\ulcorner s \urcorner^{\mathtt{I}}$, is defined dually

$$\begin{array}{lll}
\ulcorner \epsilon \urcorner^{\mathtt{I}} & = \emptyset & \\
\ulcorner s \cdot l_n \urcorner^{\mathtt{I}} & = \{n\} \cup \ulcorner s \urcorner^{\mathtt{I}} & l_n \text{ input} \\
\ulcorner s \cdot l_n \urcorner^{\mathtt{I}} & = \{n\} & l_n \text{ output}, \forall i. l_i \not\curvearrowright_{\mathsf{s}} l_n \\
\ulcorner s_1 \cdot l_i \cdot s_2 \cdot l_n \urcorner^{\mathtt{I}} & = \{i, n\} \cup \ulcorner s_1 \urcorner^{\mathtt{I}} & l_n \text{ output}, l_i \curvearrowright_{\mathsf{s}} l_n
\end{array}$$

We often confuse $\ulcorner s \urcorner^{\mathtt{O}}$ and $\ulcorner s \urcorner^{\mathtt{I}}$ with the corresponding sequences.

Let $\vdash_\phi P \triangleright A \stackrel{s}{\Longrightarrow} \vdash_\psi Q \triangleright B$. Then $s = l_1 \cdots l_n$ is *input-visible* if whenever l_i is input such that $l_j \curvearrowright l_i$, we have $j \in \ulcorner l_1 \cdots l_i \urcorner^{\mathtt{I}}$. Dually we define output-visibility. We say $\vdash_\phi P \triangleright A$ is *visible* if whenever $\vdash_\phi P \triangleright A \stackrel{s}{\Longrightarrow}$ and s is input-visible then it is output-visible.

Proposition 15. *$\vdash_\phi P \triangleright A$ is always visible.*

The proof follows [10, F15] using the switching condition, with the extra case for $\curvearrowright_{\mathsf{o}}$ (the reasoning which establishes an object biding is is visible precisely follows the output case of [10, F15]). Next, *well-bracketing* [4, 19, 24] says that later questions are always answered first, i.e. nesting of bracketing is always properly matched. Below, we call actions of mode **!** and **?** *questions* while actions of mode $\downarrow$ and $\uparrow$ are *answers*.

Let $\vdash_\phi P \triangleright A \stackrel{s}{\Longrightarrow} \vdash_\psi Q \triangleright B$ be input-visible. Then s is *well-bracketing* if, whenever $s' \cdot l \cdot t \cdot l'$ is a prefix of s and

- l is a question and
- l' is an answer, but $t \not\curvearrowright_{\mathsf{s}} l'$,

then we have $l_i \curvearrowright_{\mathsf{s}} l_j$.

Let us say $\vdash_\phi P \triangleright A$ is *well-bracketing* if whenever $\vdash_\phi P \triangleright A \stackrel{sl}{\Longrightarrow}$ is input visible, s is well-bracketing and l is output, then sl is well-bracketing. The proof of the following result again follows [11].

Proposition 16. *$\vdash_\phi P \triangleright A$ is always well-bracketing.*

Below a trace is *well-knit* if its only free subject (if any) is the in initial one. Let $\vdash_\phi P \triangleright A \stackrel{s}{\Longrightarrow}$. Then s is *legal* if it is well-knit, input-visible and well-bracketing. Define $\mathtt{trace}(\vdash_\phi P \triangleright A)$ be the set of legal traces of $\vdash_\phi P \triangleright A$. We then define $\approx_{\mathsf{seq}\forall\exists}$ by $\vdash_\phi P \approx_{\mathsf{seq}\forall\exists} Q \triangleright A \ \stackrel{\mathrm{def}}{\Leftrightarrow}\ \mathtt{trace}(\vdash_\phi P \triangleright A) = \mathtt{trace}(\vdash_\phi Q \triangleright A)$. The proof of the following uses composite transitions in the manner similar to [11].

Proposition 17. *$\approx_{\mathsf{seq}\forall\exists}$ is a typed congruence and $\approx_{\mathsf{seq}\forall\exists} \subseteq \cong_{\forall\exists}$.*

Next we introduce a useful characterisation of $\approx_{\mathsf{seq}\forall\exists}$. Let us say a legal sequence $l_1..l_n$ *matches* ϕ if either $n = 0$ or, if not, l_n is output (resp. input) iff $\phi = \mathtt{O}$ (resp. $\phi = \mathtt{I}$).

A family of typed relations $\{\mathbf{R}^{s_j}\}_j$ is a *sequential bisimulation* if, whenever $A \vdash_\phi P \, \mathbf{R}^s \, Q$, s matches l, and the following and its symmetric case hold: if $A \vdash_\phi P \stackrel{l}{\longrightarrow} B \vdash_\psi P'$ with $s \cdot \hat{l}$ legal, then $A \vdash_\phi Q \stackrel{\hat{l}}{\Longrightarrow} B \vdash_\psi Q'$ such that $B \vdash_\psi P' \, \mathbf{R}^{s \cdot \hat{l}} \, Q'$. If $A \vdash_\phi P \, \mathbf{R}^s \, Q$ for some sequential bisimulation $\mathbf{R}$, we write $A \vdash_\phi P \approx^s_{\mathsf{seq}\forall\exists} Q$.

Proposition 18. $\approx^\varepsilon_{\mathsf{seq}\forall\exists} = \approx_{\mathsf{seq}\forall\exists}$

Proof: $\approx^\varepsilon_{\mathsf{seq}\forall\exists} \subseteq \approx_{\mathsf{seq}\forall\exists}$ is immediate. For the converse, we construct a family of typed relations $\{\mathbf{R}^s\}$ as follows, starting from $\mathbf{R}^\varepsilon = \approx_{\mathsf{seq}\forall\exists}$.

$$\vdash_\phi P_1 \mathbf{R}^{s\cdot\hat{l}}_{n+1} \, P_2 \triangleright A \quad \Leftrightarrow \quad \exists Q_1, Q_2. \ \ \vdash_\psi Q_1 \mathbf{R}^s_n \, Q_2 \triangleright B \quad \vdash_\psi Q_i \triangleright B \stackrel{\hat{l}}{\Longrightarrow} \vdash_\phi P_i \triangleright A$$

Here $(i = 1, 2)$. It is easy to see that $\vdash_\phi P_1 \ \mathbf{R}^s_n \ P_2 \triangleright A$ implies that P_1 and P_2 have the same trace as far as legal sequences with the prefix s are concerned (by determinacy). We now show $\{\mathbf{R}^s\}$ is a bisimulation. Suppose $\vdash_\phi P_1 \mathbf{R}^s_n P_2 \triangleright A$ and $\vdash_\phi P_1 \triangleright A \stackrel{l}{\longrightarrow} \vdash_\psi P_1' \triangleright B$ such that $s \cdot l$ is legal. If $l = \tau$ we simulate this by the non-transition of $\vdash_\phi P_2 \triangleright A$ and the result is in $\mathbf{R}^s$ again. Otherwise this is simulated by $\vdash_\phi P_2 \triangleright A \stackrel{l}{\Longrightarrow} \vdash_\psi P_2' \triangleright B$ and the result is in $\mathbf{R}^{sl}$ by definition, hence $\mathbf{R}_0 \subseteq \approx^\varepsilon_{\mathsf{seq}\forall\exists}$. □

Proposition 19. *If $\vdash_\phi P_1 \approx^s_{\mathsf{seq}\forall\exists} P_2 \triangleright A$ and $\vdash_\phi P_i \triangleright A \stackrel{l}{\longrightarrow} \vdash_\psi P_i' \triangleright B$ $(i = 1, 2)$ such that $s \cdot l$ is legal, then $\vdash_\psi P_1' \approx^{sl}_{\mathsf{seq}\forall\exists} P_2' \triangleright B$.*

5.5 Innocence

We can now introduces *innocence*, the fundamental property of generic transition of polymorphic affine processes. Let $\vdash_\phi P \triangleright A$ with t legal. The proof of the following result precisely follows [11].

Lemma 13. (permutation) *Let* $\vdash_{\mathtt{I}} P \triangleright A \stackrel{l_1 \cdot l_2 \cdot l_3 \cdot l_4}{\Longrightarrow} \vdash_{\mathtt{I}} Q \triangleright B$ *such that* $l_1 \not\curvearrowright l_4$ *and* $l_2 \not\curvearrowright l_3$*. Then* $\vdash_{\mathtt{I}} P \triangleright A \stackrel{l_3 \cdot l_4 \cdot l_1 \cdot l_2}{\Longrightarrow} \vdash_{\mathtt{I}} Q \triangleright B$*.*

By the above lemma and visibility, we can transform any transition of form $\vdash_\phi P \triangleright A \stackrel{sl}{\Longrightarrow}$, with l output, to $\vdash_\phi P \triangleright A \stackrel{tl}{\Longrightarrow}$ where $t = \ulcorner s \urcorner^0$. Since an output is always unique, we can conclude:

Proposition 20. (innocence) *Let* $\vdash_\psi P \triangleright A \stackrel{s_1 l_1}{\Longrightarrow}$ *and* $\vdash_\psi P \triangleright A \stackrel{s_2}{\Longrightarrow}$ *such that:* (1) *both* s_1 *and* s_2 *are legal;* (2) l_1 *is an output; and* (3) $\ulcorner s_1 \urcorner^0 \equiv_\alpha \ulcorner s_2 \urcorner^0$*. Then we have* $\vdash_\psi P \triangleright A \stackrel{s_2 l_2}{\Longrightarrow}$ *such that* $\ulcorner s_1 \urcorner^0 \cdot l_1 \equiv_\alpha \ulcorner s_2 \urcorner^0 \cdot l_2$*.*

Remark 3. The equalities $\ulcorner s_1 \urcorner^0 \cdot l_1 \equiv_\alpha \ulcorner s_2 \urcorner^0 \cdot l_2$ and $\ulcorner s_1 \urcorner^0 \equiv_\alpha \ulcorner s_2 \urcorner^0$ include information about bindings of free objects, which differs from the first-order case [11].

By Proposition 20, the behaviour of a process can be regarded as a function mapping each output view ending with an input to the next action [11, 24]. Such functions are called *generic innocent function*, or often simply *innocent function*. By Corollary 17 this induces a sound congruence w.r.t. $\cong_{\forall\exists}$. An innocent function is *finite* if its function representation is finite in the sense of set theory (i.e. if it can be described by a finite number of tuples). If $\vdash_\phi P \triangleright A$, then $\mathsf{inn}(\vdash_\phi P \triangleright A)$ is the finite innocent function induced by P. P is *finite* if $\mathsf{inn}(P)$ is finite innocent. Innocent functions of linear polymorphic processes are *total* (but not necessarily finite) in the sense that a legal input is always followed by an output. An innocent function is of type (ϕ, A) if it coincides with $\mathsf{inn}(\vdash_\phi P \triangleright A)$ for some $\vdash_\phi P \triangleright A$.

Polymorphic linear processes also enjoy finite representability, i.e. finite generic innocent functions are always representable by typed processes. Following [11], we only consider well-knit traces under a typing which is *well-formed* in the sense that: (1) it is the result of composing several connected types, and (2) it does not contain more than one free linear type. We hereafter identify a set of well-knit traces which define a generic innocent function, with the innocent function itself. By the construction it suffices to consider only those well-knit traces which are output views.

In the next section we will need to know that replacing a process by its maximally copycatted counterpart does not affect the induced innocent function.

Lemma 14. *For every* $\vdash_\phi P \triangleright A$*, there's a maximally copycatted* Q *such that* $\mathsf{inn}(\vdash_\phi P \triangleright A) = \mathsf{inn}(\vdash_\phi Q \triangleright A)$*.*

Proof. Straightforward. $\square$

6 Reasoning Examples

This section demonstrates that equational reasoning based on combining $\mapsto$ and $\stackrel{l}{\longrightarrow}$ gives powerful reasoning tools for generic processes.

Examples using Generic Transitions. We have already seen the behaviour of $S \stackrel{def}{=} \overline{x}(yz)(\mathsf{t}\langle y\rangle | z(b').\overline{e}(b)\mathsf{not}\langle bb'\rangle)$ under the typing $x : \exists \mathrm{X}.(\mathrm{X}(\overline{\mathrm{X}})^{\downarrow})^{?} \cdot e : (\mathbb{B})^{\uparrow}$ in Section 2. If x is typed $(\mathbb{B}(\overline{\mathbb{B}})^{\downarrow})^{?}$ we have the following transitions, assuming $B \stackrel{def}{=} x : (\mathbb{B}(\overline{\mathbb{B}})^{\downarrow})^{?} \cdot e : (\mathbb{B})^{\uparrow}$,

$$\begin{aligned} \vdash_{\mathtt{O}} S \triangleright B &\stackrel{\overline{x}(yz)}{\longrightarrow} \vdash_{\mathtt{I}} \mathsf{t}\langle y\rangle | z(b').\overline{e}(b)\mathsf{not}\langle bb'\rangle \triangleright B \cdot y : \mathbb{B} \cdot z : (\overline{\mathbb{B}})^{\downarrow} \\ &\stackrel{z(b')}{\longrightarrow} \vdash_{\mathtt{O}} \mathsf{t}\langle y\rangle | \overline{e}(b)\mathsf{not}\langle bb'\rangle \triangleright B \cdot y : \mathbb{B} \cdot b' : \overline{\mathbb{B}} \end{aligned}$$

where we can only use a bound input at the second transition. Note also that y is typed with a non-opaque type, so that an action at y is possible.

Next we look at a transition sequence of $\mathsf{id}\langle x\rangle$, first with universal abstraction.

$$\begin{aligned} \vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle \triangleright x : \mathbb{I} &\stackrel{x(yz)}{\longrightarrow} \vdash_{\mathtt{O}} \mathsf{id}\langle x\rangle | \overline{z}\langle y\rangle \triangleright x : \mathbb{I} \cdot y : \overline{\mathrm{X}}^{\forall} \cdot z : (\mathrm{X}^{\forall})^{\uparrow} \\ &\stackrel{\overline{z}\langle y\rangle}{\longrightarrow} \vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle | 0 \triangleright x : \mathbb{I} \cdot y : \overline{\mathrm{X}}^{\forall} \end{aligned}$$

Now consider an instance of the monomorphic identity which is the same untyped process $\mathsf{id}\langle x\rangle$ under a different typing, $x : (\overline{\mathbb{B}}(\mathbb{B})^{\uparrow})^{!}$. The process has the following initial transitions.

$$\begin{aligned} \vdash_{\mathtt{I}} \mathsf{id}\langle x\rangle \triangleright x : (\overline{\mathbb{B}}(\mathbb{B})^{\uparrow})^{!} &\stackrel{x(yz)}{\longrightarrow} \vdash_{\mathtt{O}} \mathsf{id}\langle x\rangle | \overline{z}\langle y\rangle \triangleright x : (\overline{\mathbb{B}}(\mathbb{B})^{\uparrow})^{!} \cdot y : \overline{\mathbb{B}} \cdot z : (\mathbb{B})^{\uparrow} \\ &\stackrel{\overline{z}(y')}{\longrightarrow} \vdash_{\mathtt{O}} \mathsf{id}\langle x\rangle | [y' \to y]^{\mathbb{B}} \triangleright x : (\overline{\mathbb{B}}(\mathbb{B})^{\uparrow})^{!} \cdot y : \overline{\mathbb{B}} \cdot y' : \mathbb{B} \end{aligned}$$

Here we do not allow free output via z, since there is no generically typed name. As a result, a copy-cat remains in the configuration.

Inhabitation Results. We now show how easy inhabitation results are to come by with generic transitions. Inhabitation seeks to characterise the processes typable by a particular type. We prove:

Lemma 15. (inhabitation of $\mathbb{I}$) $\vdash_{\mathtt{I}} P \triangleright x : \mathbb{I}$ *implies* $P \approx_{\forall\exists} \mathsf{id}\langle x\rangle$.

Proof. Let $\vdash_{\mathtt{I}} P \triangleright x : \mathbb{I}$. Then we have

$$\vdash_{\mathtt{I}} P \triangleright x : \mathbb{I} \stackrel{x(yz)}{\longrightarrow} \vdash_{\mathtt{O}} P' \triangleright x : \mathbb{I}, y : \mathrm{X}^{\forall}, z : (\mathrm{X}^{\forall})^{\uparrow}$$

By inspecting the action type, if P' ever has an output, it can only be $\overline{z}\langle y\rangle$, in which case $P \approx_{\forall\exists} \mathsf{id}\langle x\rangle$. □

Similarly we can check directly that $x : \mathbb{B}$ is inhabited by $\mathsf{t}\langle x\rangle$ and $\mathsf{f}\langle x\rangle$ only, using the transition relation instead of using $\mapsto$ as in Lemma 7.

We can also use this lemma to reason about more complex terms by combining $\mapsto$ and $\stackrel{l}{\longrightarrow}$. In Section 2, we have seen the behaviour of the process

$$S \stackrel{def}{=} \overline{x}(yz)(\mathsf{t}\langle y\rangle|z(w).\overline{e}(b)\mathsf{not}\langle bw\rangle) \text{ under } x:\overline{\mathbb{I}}, e:(\mathbb{B})^{\uparrow}.$$

We show S and $S' \stackrel{def}{=} \overline{e}(b)\mathsf{f}\langle b\rangle$ are contextually congruent. Since S and S' have different visible traces, the use of some extensionality principle is essential. Here it suffices to show $(\boldsymbol{\nu}\, x)(S|P) \cong_{\forall\exists} (\boldsymbol{\nu}\, x)(S'|P)$ for each $\vdash_{\mathtt{I}} P \triangleright x : \mathbb{I}$. But if $\vdash_{\mathtt{I}} P \triangleright x : \mathbb{I}$ then $P \cong_{\forall\exists} \mathsf{id}\langle x\rangle$ by the above inhabitation result. Using extended reduction we can check $(\boldsymbol{\nu}\, x)(S|P) \cong_{\forall\exists} (\boldsymbol{\nu}\, x)(S|\mathsf{id}\langle x\rangle) \mapsto^{+} S'$ and $(\boldsymbol{\nu}\, x)(S'|P) \cong_{\forall\exists} (\boldsymbol{\nu}\, x)(S'|\mathsf{id}\langle x\rangle) \mapsto^{+} S'$. Since $\mapsto^{+}\subseteq\cong_{\forall\exists}$, we have $(\boldsymbol{\nu}\, x)(S|P) \cong_{\forall\exists} S' \cong_{\forall\exists} (\boldsymbol{\nu}\, x)(S'|P)$, hence done.

Boolean ADTs. Next is simple use of transition relations for reasoning about abstract data types of opaque booleans (similar to those discussed in [37, 40]). The data type should export "flip", or negation and a read operation (the latter turns an opaque boolean to a concrete boolean). Two simple implementations in the λ-calculus with records are:

$$\begin{aligned} M &\stackrel{def}{=} \mathtt{pack\ bool}\ \{\mathtt{bit} = \mathsf{T},\ \mathtt{flip} = \lambda x{:}\mathtt{bool}.\neg x,\ \mathtt{read} = \lambda x{:}\mathtt{bool}.x\ \}\ \mathtt{as}\ \mathit{bool} \\ M' &\stackrel{def}{=} \mathtt{pack\ bool}\ \{\mathtt{bit} = \mathsf{F},\ \mathtt{flip} = \lambda x{:}\mathtt{bool}.\neg x,\ \mathtt{read} = \lambda x{:}\mathtt{bool}.\neg x\}\ \mathtt{as}\ \mathit{bool} \end{aligned}$$

where $\mathit{bool} \stackrel{def}{=} \exists \mathrm{X}.\{\mathtt{bit} : \mathrm{X},\ \mathtt{flip} : \mathrm{X} \to \mathrm{X}, \mathtt{read} : \mathrm{X} \to\ \mathtt{bool}\}$. M and M' can be encoded as (using a CBV translation of products, cf. [48]):

$$\mathtt{bool}\langle u\rangle \stackrel{def}{=} \overline{u}(m_1m_2m_3)(Q_1|Q_2|Q_3) \qquad \mathtt{bool}'\langle u\rangle \stackrel{def}{=} \overline{u}(m_1m_2m_3)(Q_1'|Q_2'|Q_3')$$

where

$$\begin{array}{lll} Q_1 \stackrel{def}{=} \mathsf{t}\langle m_1\rangle & Q_2 \stackrel{def}{=} !m_2(bz).\overline{z}(b')\mathsf{not}\langle b'b\rangle & Q_3 \stackrel{def}{=} !m_3(bz).\overline{z}\langle b\rangle \\ Q_1' \stackrel{def}{=} \mathsf{f}\langle m_1\rangle & Q_2' \equiv Q_2 & Q_3' \stackrel{def}{=} !m_3(bz).\overline{z}(b')\mathsf{not}\langle b'b\rangle \end{array}$$

We can easily check that these processes are typable under $u : \exists \mathrm{X}.\mathcal{B}[\mathrm{X}]$, where $\mathcal{B}[\mathrm{X}] \stackrel{def}{=} (\mathrm{X}(\overline{\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}(\overline{\mathrm{X}}(\mathbb{B})^{\uparrow})^{!})^{\uparrow}$.

We now show $\vdash_{\mathtt{I}} \mathtt{bool}\langle u\rangle \approx_{\forall\exists} \mathtt{bool}'\langle u\rangle \triangleright x : \exists \mathrm{X}.\mathcal{B}[\mathrm{X}]$. For a proof, we show these agents are in a trace equivalence up to $\mapsto$, then use Proposition 17. First we analyse the transition relations from $\mathtt{bool}\langle u\rangle$. Let $R = Q_1 \mid Q_2 \mid Q_3$ and $R' = Q_1' \mid Q_2' \mid Q_3'$. Then after output transition $\overline{u}\langle(\boldsymbol{\nu}\, m_2m_3)m_1m_2m_3\rangle$, we have:

$$\begin{aligned} \vdash_{\mathtt{I}} (\boldsymbol{\nu}\, m_2'm_3')(R\{m_2'/m_2\}\{m_3'/m_3\} \mid [m_2 \to m_2'] \mid [m_3 \to m_3']) \ &= \ R_1 \\ \triangleright m_1 : \mathrm{X}, m_2 : (\overline{\mathrm{X}}(\mathrm{X})^{\uparrow})^{!}, m_3 : (\overline{\mathrm{X}}(\mathbb{B})^{\uparrow})^{!} & \end{aligned}$$

Similarly for R'. Since X is an existential type variable, via m_2, R_1 can only input m_1 as the first argument. Similarly, by analysing types, R_1 has the following sequential trace:

$$R_1 \stackrel{m_2\langle(\nu c)m_1c\rangle}{\Longrightarrow} R_2 \stackrel{\overline{c}(b)}{\Longrightarrow} R_3 \stackrel{m_3\langle(\nu e)be\rangle}{\Longrightarrow} R_4$$

Note c in R_2 has type $c{:}(\mathrm{X})^{\uparrow}$. Hence when R_2 exports bound name b via c, the same name b should be returned via m_3 as the first argument [because b has type $\overline{\mathrm{X}}$, while m_3 has type $(\overline{\mathrm{X}}(\mathbb{B})^{\uparrow})^{!}$ in R_3]. We now have:

$$R_4 \mapsto^{*} \overline{e}(b)\mathsf{not}\langle bm_1\rangle \mid \mathsf{t}\langle m_1\rangle \mid Q_2 \mid Q_3 \mapsto^{*} \overline{e}(b)\mathsf{f}\langle b\rangle \mid \mathsf{t}\langle m_1\rangle \mid Q_2 \mid Q_3$$

By the exactly same trace from R', we can obtain:

$$R'_4 \mapsto^* (\nu b')(\overline{e}(b)\mathsf{not}\langle bb'\rangle \mid \mathsf{not}\langle b'm_1\rangle \mid \mathsf{f}\langle m_1\rangle) \mid Q'_2 \mid Q'_3 \mapsto^* \overline{e}(b)\mathsf{f}\langle b\rangle \mid \mathsf{f}\langle m_1\rangle \mid Q'_2 \mid Q'_3$$

We have the essentially same reasoning when R or R' starts from the input action at m_3, and they reduces into the same normal forms as the above. Thus we conclude $\mathtt{bool}\langle u\rangle \approx_{\forall\exists} \mathtt{bool}'\langle u\rangle$.

7 Full Abstraction of System F

This section embeds System F fully abstractly with respect to Moggi-Statman's maximal consistent theory (which is the standard maximum contextual congruence for the formalism). There are two main interests. First it shows that the duality-based typing can precisely embed existing functional formalisms, connecting the world of typed functions and the world of typed processes as well as opening the potential to use the generic π-calculus for a meta-language. Second the proof highlights one of the deep aspects of generic behaviour, information hiding by existential types (equivalently by contravariant universal types). The encoding and the result smoothly extend to inclusion of standard data types or nontermination.

We briefly summarise the syntax of $\lambda^{\forall}$ with types (α, β, ...) and preterms (M, N, ...) given by the following grammar.

$$\alpha \ ::= \ \mathrm{X} \mid \alpha \Rightarrow \beta \mid \forall \mathrm{X}.\alpha \qquad M \ ::= \ x \ \mid \ \lambda x^{\alpha}.M \ \mid \ MN \ \mid \ \Lambda \mathrm{X}.M \ \mid \ M\beta.$$

We often omit type annotation on bound names. The standard reduction is generated from $(\lambda x^{\alpha}.M)N \longrightarrow M\{N/x\}$ (which we call β_v-reduction) and $(\Lambda \mathrm{X}.M)\beta \longrightarrow M\{\beta/\mathrm{X}\}$ (which we call type instantiation), as well as by compatible closure. $M \Downarrow N$ is defined as: $M \longrightarrow^* N \not\longrightarrow$.

We list the typing rules for reference (we use bases with type variable declarations for clarity). We use the judgement $E \vdash \alpha$ which is true iff all type variables in α are declared in E. Well-formedness of E is assumed, i.e. a type variable is declared in E before it is assigned to a variable.

$$\frac{}{E, x : \alpha \vdash x : \alpha} \qquad \frac{E, x : \alpha \vdash M : \beta}{E \vdash \lambda x{:}\alpha.M : \alpha \Rightarrow \beta} \qquad \frac{E \vdash M : \alpha \Rightarrow \beta \quad E \vdash N : \alpha}{E \vdash MN : \beta}$$

$$\frac{E, \vdash M : \alpha \quad \mathrm{X} \notin \mathsf{ftv}(E)}{E \vdash \Lambda \mathrm{X}.M : \forall \mathrm{X}.\alpha} \qquad \frac{E \vdash M : \forall \mathrm{X}.\alpha \quad E \vdash \beta}{E \vdash M\beta : \alpha\{\beta/\mathrm{X}\}}$$

Let $\mathbb{B}_\lambda \stackrel{\text{def}}{=} \forall \mathrm{X}.\mathrm{X} \Rightarrow (\mathrm{X} \Rightarrow \mathrm{X})$, $\mathsf{T} = \Lambda \mathrm{X}.\lambda x^{\mathrm{X}}.\lambda y^{\mathrm{X}}.x$ and $\mathsf{F} = \Lambda \mathrm{X}.\lambda x^{\mathrm{X}}.\lambda y^{\mathrm{X}}.y$. Then the relation $\cong_\forall$ [34] is the largest congruence satisfying: $\forall M_{1,2} : \mathbb{B}_\lambda.\ M_1 \cong_\forall M_2 \ \Leftrightarrow \ (M_1 \Downarrow \mathsf{T} \Leftrightarrow M_2 \Downarrow \mathsf{T})$. It can be shown that T and F are essentially the only inhabitants of $\mathbb{B}_\lambda$.

Proposition 21. *Let* $\vdash M : \mathbb{B}_\lambda$ *and* $M \not\longrightarrow$. *Then* $M \equiv_\alpha \mathsf{T}$ *or* $M \equiv_\alpha \mathsf{F}$.

7.1 Encoding and Adequacy

Because it simplifies the definability argument, we use Milner's untyped call-by-value encoding (CBV) [28], which only differs from Turner's translation [45] in the treatment of quantification. The target calculus is that of Section 3, but for reasoning with labelled transitions, we use the extension with annotated type variables of Section 5 (cf. [20]). It is easy to see that any encoding induces an infinite number of other encodings by simply adding more and more layers of indirection. As we shall sketch later, call-by-name (CBN) or other reasonable encodings (for example that of [11]) induce isomorphic translation of types, so the full abstraction result immediately extends to these encodings.

The mappings $\alpha^\bullet$ (for types) and $[\![M]\!]_u$ (for terms) are as follows, assuming newly introduced names are chosen to be fresh.

$$
\begin{array}{l}
\alpha^\bullet \stackrel{def}{=} (\alpha^\circ)^\uparrow \quad \mathrm{X}^\circ \stackrel{def}{=} \mathrm{X}^! \quad (\alpha \Rightarrow \beta)^\circ \stackrel{def}{=} (\overline{\alpha^\circ}\beta^\bullet)^! \quad (\forall \mathrm{X}.\alpha)^\circ \stackrel{def}{=} \forall \mathrm{X}.(\alpha^\bullet)^! \\
[\![x]\!]_u \stackrel{def}{=} \overline{u}\langle x\rangle \qquad [\![\lambda x^\alpha.M]\!]_u \stackrel{def}{=} \overline{u}(m)!m(xz).[\![M]\!]_z \\
[\![MN]\!]_u \stackrel{def}{=} (\boldsymbol{\nu}\, m)([\![M]\!]_m \mid m(a).(\boldsymbol{\nu}\, n)([\![N]\!]_n \mid n(b).\overline{a}\langle bu\rangle)) \\
[\![\Lambda \mathrm{X}.M]\!]_u \stackrel{def}{=} \overline{u}(a)!a(m).[\![M]\!]_m \qquad [\![M\beta]\!]_u \stackrel{def}{=} (\nu m)([\![M]\!]_m \mid m(v).\overline{v}u)
\end{array}
$$

A base is encoded as (ignoring type variable declarations): $\emptyset^\circ = \emptyset$ and $(E \cdot y : \alpha)^\circ = E^\circ, y : \overline{\alpha^\circ}$.

An alternative, more efficient encoding for universal abstraction would be to have $(\forall \mathrm{X}.\alpha)^\circ = \forall \mathrm{X}.\alpha^\circ$. Unfortunately $\forall \mathrm{X}.\mathrm{X}$ is not a well-defined type in our system, so such an encoding is not typable. It is fairly straightforward to adapt our calculus to make $\forall \mathrm{X}.\mathrm{X}$ an admissible type, at the cost at introducing an additional *generic wire* construct, but for simplicity we have opted to stay with the most straightforward typing system.

Proposition 22. *$E \vdash M : \alpha$ implies $\vdash_{\mathsf{0}} [\![M]\!]_u \triangleright u : \alpha^\bullet, E^\circ$.*

Proof. By straightforward induction on the derivation of the source term. □

Lemma 16. *Let $x, y \neq u$ and $E \vdash M : \alpha$. Then $[\![M]\!]_u\{x/y\} = [\![M\{x/y\}]\!]_u$.*

Proof. By an easy induction on $E \vdash M : \alpha$. □

Proposition 23. *Let $E \vdash M : \alpha$ and $M \longrightarrow N$. Then $[\![M]\!]_u \mapsto^+ [\![N]\!]_u$.*

Proof. Straightforward by induction on the derivation of $E \vdash M : \alpha$. As an example, we calculate β-reduction, i.e. $[\![(\lambda x.M)V]\!]_u$. We start with $V = y$.

$$
\begin{array}{rl}
[\![(\lambda x.M)y]\!]_u & \equiv (\nu l)(\overline{l}(a)!a(xm).[\![M]\!]_m \mid l(a).(\boldsymbol{\nu}\, v)([\![y]\!]_v \mid v(x).\overline{a}\langle xu\rangle)) \\
& \mapsto (\nu a)(!a(xm).[\![M]\!]_m \mid (\boldsymbol{\nu}\, v)([\![y]\!]_v \mid v(x).\overline{a}\langle xu\rangle)) \\
& \mapsto (\nu a)(!a(xm).[\![M]\!]_m) \mid (\boldsymbol{\nu}\, v)([\![y]\!]_v \mid v(x).[\![M]\!]_u) \\
& \mapsto (\boldsymbol{\nu}\, v)([\![y]\!]_v \mid v(x).[\![M]\!]_u) \\
& \equiv (\boldsymbol{\nu}\, v)(\overline{v}y \mid v(x).[\![M]\!]_u) \\
& \mapsto [\![M\{y/x\}]\!]_u
\end{array}
$$

There is another possibility, $V = (\lambda y.N)$.

$$\begin{aligned}
[\![(\lambda x.M)V]\!]_u &\equiv (\nu l)(\bar{l}(a)!a(xm).[\![M]\!]_m \mid l(a).(\boldsymbol{\nu}\, v)([\![V]\!]_v \mid v(x).\overline{a}\langle xu\rangle))\\
&\mapsto (\nu a)(!a(xm).[\![M]\!]_m \mid (\boldsymbol{\nu}\, v)([\![V]\!]_v \mid v(x).\overline{a}\langle xu\rangle))\\
&\equiv (\nu a)(!a(xm).[\![M]\!]_m \mid (\boldsymbol{\nu}\, v)(\overline{v}(b)!b(yn).[\![N]\!]_n \mid v(x).\overline{a}\langle xu\rangle))\\
&\mapsto (\nu a)(!a(xm).[\![M]\!]_m \mid (\nu b)(!b(yn).[\![N]\!]_n \mid \overline{a}\langle bu\rangle))\\
&\mapsto (\nu ab)(!a(xm).[\![M]\!]_m \mid [\![M\{b/x\}]\!]_u \mid !b(yn).[\![N]\!]_n \mid \overline{a}\langle bu\rangle))\\
&\mapsto (\nu b)([\![M\{b/x\}]\!]_u \mid !b(yn).[\![N]\!]_n)
\end{aligned}$$

It is easy to demonstrate that

$$(\nu b)([\![M\{b/x\}]\!]_u \mid !b(yn).[\![N]\!]_n) \mapsto\!\!\!\!\!\rightarrow [\![M\{V/x\}]\!]_u,$$

completing the proof. $\square$

Since any infinite reduction path in CBV System F contains infinitely many β_v-reductions, this gives its strong normalisability (hence that of call-by-name reduction). By noting $[\![\mathsf{T}]\!]_u \in \mathsf{NF}_e$, we obtain:

Proposition 24. (computational adequacy) *Let* $\vdash M : \mathbb{B}_\lambda$*. Then* $M \Downarrow \mathsf{T}$ *if and only if* $[\![M : \mathbb{B}_\lambda]\!]_u \Downarrow_e [\![\mathsf{T}]\!]_u$*.*

Proof. From Proposition 23 ($\Rightarrow$) follows immediately. For the reverse implication, let $[\![M : \mathbb{B}_\lambda]\!]_u \Downarrow_e [\![\mathsf{T}]\!]_u$ and $M \Downarrow N$. By Proposition 21 either $N \equiv \mathsf{T}$ or $N \equiv \mathsf{F}$. By Proposition 23 also $[\![M]\!]_u \Downarrow [\![N]\!]_u$. Then $[\![N]\!]_u \equiv [\![\mathsf{T}]\!]_u$ by Lemma 9. This is clearly only possible if $N \equiv \mathsf{T}$. $\square$

Corollary 1. (equational soundness) *Let* $\vdash M_{1,2} : \alpha$*. Then* $\vdash_{\mathtt{I}} [\![M_1]\!]_u \cong_{\forall\exists} [\![M_2]\!]_u \triangleright u : \alpha^\bullet$ *implies* $M_1 \cong_\forall M_2 : \alpha$*.*

Proof. The proof is straightforward. Let $M\ \mathbf{R}\ N$ iff $\vdash_{\mathtt{I}} [\![M]\!]_u \cong_{\forall\exists} [\![N]\!]_u \triangleright u : \alpha^\circ$. We show that $\mathbf{R}$ is a sound Moggi-Statman Theory. Consistency and congruency of $\mathbf{R}$ follows from that of $\cong_{\forall\exists}$. For reduction closure, let $M \longrightarrow M'$ and $[\![M]\!]_u \cong_{\forall\exists} [\![N]\!]_u$. Then $[\![M]\!]_u \mapsto\!\!\!\!\!\rightarrow [\![M']\!]_u$ (Proposition 23). But then $[\![M']\!]_u \cong_{\forall\exists} [\![N]\!]_u$ by determinacy of $\Downarrow_e$. Finally, preservation of barbs is immediately by Proposition 24. $\square$

7.2 Definability

We shall now complete the proof of full abstraction. We begin by explaining why the key step, definability, is tricky.

Why is Definability Difficult to Establish? Definability means that for every process $\vdash_{\mathtt{0}} P \triangleright u : \alpha^{\bullet}, \overline{E^{\circ}}$ of translated System F type, we can find a term $E \vdash M : \alpha$ such that $P \cong_{\forall\exists} [\![M]\!]_u$. In the first-order case, we can prove the corresponding theorem by induction on the derivation of P's typing judgement [48]. Let's see what happens if we try to do this naively in the current setting. Consider a process $\vdash_{\mathtt{0}} \overline{x}(a)a(b).P \triangleright u : \alpha^{\bullet}, x : \overline{(\forall \mathrm{X}.\beta)^{\circ}}, \overline{E^{\circ}}$ where $\vdash_{\mathtt{0}} P \triangleright u : \alpha^{\bullet}, b : \overline{\beta^{\circ}\{\mathrm{X}^{\exists}/\mathrm{X}^{\forall}\}}, x : \overline{(\forall \mathrm{X}.\beta)^{\circ}}, \overline{E^{\circ}}$. If β freely contains X we have a problem: P may not be typable by the translation of System F types. Consider for example $\beta = (\mathrm{X} \Rightarrow \mathrm{X})$. In this case

$$P \equiv \overline{b}(cd)(!c(\vec{v}).Q \mid d(v).S), \qquad \vdash_{\mathtt{0}} Q \triangleright \vec{v} : \vec{\tau}, \overline{F^{\circ}}, \qquad \vdash_{\mathtt{0}} R \triangleright v : (\vec{\overline{\tau}})^{\downarrow}, \overline{F^{\circ}}$$

where $\vec{\tau}$ is a vector of *arbitrary* length, precluding decompilation into System F types. Now how do we apply the inductive hypothesis? On top of this, it is also far from obvious how to order typing judgements to ensure things like $u : \alpha^{\bullet}, b : \overline{\beta^{\circ}\{\mathrm{X}^{\exists}/\mathrm{X}^{\forall}\}}, x : \overline{(\forall \mathrm{X}.\beta)^{\circ}}, \overline{E^{\circ}}$ being 'smaller' than $u : \alpha^{\bullet}, x : \overline{(\forall \mathrm{X}.\beta)^{\circ}}, \overline{E^{\circ}}$. But that would be necessary for a well-founded induction.

We overcome these problems by abandoning induction on typing derivations for induction on the size of the corresponding innocent functions, cf. [4, 11, 24]. While this itself is a standard idea for definability of affine (nonterminating) pure functions, the corresponding proof technique in the present setting involves another twist, since we should deal with free inputs, which do not exist in the standard first-order setting, either linear or affine. The proof technique to deal with free inputs indeed visibly reflect the information hiding in the behaviour of generic processes induced by typing, using the fact that the environment can only pass-on or drop $\mathrm{X}^{\exists}$-typed names. That is, if we replace $\mathrm{X}^{\exists}$ with any other type, the environment wouldn't notice. So we have to find a "representative" τ for $\mathrm{X}^{\exists}$ to apply the inductive hypothesis such that substituting τ for $\mathrm{X}^{\exists}$ does not change observable behaviour. We can then use ($\exists$-VAR) to reconstruct the original type. To see how one could find such a type, let's have a look at the following example:

$$\vdash_{\mathtt{I}} !y(uvw).Q \mid !z(uvw).Q \mid r(v).\overline{v}(abc)R \triangleright y : \mathrm{X}^{\exists}, z : \mathrm{X}^{\exists}, r : (\overline{\mathrm{X}^{\exists}})^{\downarrow}, \ldots$$

where y as well as z had already been exported to the environment. Call this process P'. It arises, up to contextual congruence, as a $\xrightarrow{l}$-descendant of processes like P above. This process has two transitions:

$$P' \xrightarrow{r\langle y\rangle} !y(uvw).Q \mid !z(uvw).Q \mid \overline{y}(abc)R$$
$$P' \xrightarrow{r\langle z\rangle} !y(uvw).Q \mid !z(uvw).Q \mid \overline{z}(abc)R$$

This suggest that the output of a $\mathrm{X}^{\exists}$-typed name and its subsequent reimport allows the environment to choose between different alternatives presented by the process. Of course System F can also implement choices, for example by using $[n]_{\lambda}$ which was defined to be $\forall \mathrm{X}.(\underbrace{\mathrm{X} \Rightarrow \cdots \mathrm{X}}_{n} \Rightarrow \mathrm{X})$. Could the following transformation solve the problem of inappropriate $\mathrm{X}^{\exists}$-typed names in the induction?

– Replace $\mathrm{X}^{\exists}$ by $[n]_\lambda$, where n is the number of $\mathrm{X}^{\exists}$-typed free names in the environment before reimport.
– Transform processes $P \stackrel{def}{=} C[\ !y_i(\vec{v}.P_i)\ ]_{i=1}^{m}[\ \overline{z_j}\langle\vec{a_j}\rangle\ ]_{j=1}^{n}$, where the y_i and z_j exhaust all the $\mathrm{X}^{\exists}$-typed names, to

$$\mathsf{trans}(P) \stackrel{def}{=} C[\ \langle\!\langle \lambda x_1...x_m.x_i\rangle\!\rangle_{y_i}\ ]_{i=1}^{m}[\ \overline{z_j}(a_1r_1)r_1(s)...r_m(s).(\overline{s}\langle\vec{a_j}\rangle \mid \Pi_{i=1}^{m}!a_i(\vec{v}).P_i)\ ]_{i=1}^{n}.$$

Here $[\![\lambda x_1...x_n.x_i]\!]_u = \overline{y}(a)\langle\!\langle\lambda x_1...x_n.x_i\rangle\!\rangle_a$.

Clearly, if $\vdash_\phi P \triangleright A$ is typable using $\mathrm{X}^{\exists}$-typed names, $\vdash_\phi \mathsf{trans}(P) \triangleright A\{[n]_\lambda^\circ/\mathrm{X}^{\exists}\}$. What this transformation does is take away the environment's ability to choose one of P's servers directly. Instead we let it decide among alternative choosers of translated System F type. It can be shown that this does not change the processes visible transitions, assuming well-typed observers.

To find out a concrete type corresponding to an existential type variable, we need to know out of how many alternatives the environment can choose. Since we are going to work solely on finite innocent functions, the number of exported names of that type are always finite (in general these names may not be all possible visible names which can be inputted as each point: however this does not matter for our construction, since having redundant choices are semantically harmless).

Before going into the technical development, we show another example where existential typed names are exported to be used for a re-import on later occasions. The example is essentially Church numerals, which are typed as $\forall\mathrm{X}.((\mathrm{X} \Rightarrow \mathrm{X}) \Rightarrow (\mathrm{X} \Rightarrow \mathrm{X}))$. If $\vdash_0 P \triangleright u : \alpha^\bullet, x : \overline{\mathbb{CN}^\circ}, \overline{E^\circ}$, then

$$P \equiv \overline{x}(a)a(b)\overline{b}(cr)(\ !c(vm).Q \mid r(d).\overline{d}(es)(!e(vn).R \mid s(t).S)\ ).$$

Omitting details, P has the following transition sequence.

$$P \xrightarrow{\overline{x}(a)}\xrightarrow{a(b)}\xrightarrow{\overline{b}(cr)}\xrightarrow{r(d)}\xrightarrow{\overline{d}(ew)} C[\ !c(vm).Q \mid !e(vn).R \mid s(t).S\ ]$$

At this point, the environment has c and e as free names and may feed the latter as an argument to the former:

$$C[\,!c(vm).Q \mid !e(vn).R \mid s(t).S\,] \stackrel{c\langle(\nu f)ef\rangle}{\longrightarrow} C[\,!c(vm).Q \mid !e(vn).R \mid s(t).S \mid Q\{fe/vm\}\,].$$

In this case it is possible we can have unbounded number of existentially typed names if we consider possibly non-finite innocent functions (processes corresponding to polymorphic Church numerals are basic examples which may have non-finite innocent functions: for example, a successor is such).

Proving Definability. We are now ready to tackle definability and begin with some helpful facts.

Lemma 17. *If* $\vdash_0 P \triangleright A$ *then* $\vdash_0 P \triangleright A \stackrel{l}{\Longrightarrow} \cdots$ *for some visible output* l.

Lemma 18. *Let* $\vdash_{\mathsf{O}} P \triangleright A \stackrel{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}{\Longrightarrow} \vdash_{\mathsf{I}} Q \triangleright B$ *where* $\vec{y}$ *is a non-empty vector of names. Then* $\vdash_{\mathsf{I}} Q \triangleright B \stackrel{l}{\longrightarrow} \cdots$ *for some visible input* l*.*

Lemma 19. *Let* $P \not\mapsto$ *and* $\vdash_{\mathsf{O}} P \triangleright x : (\mathrm{X}^{\forall})^{\uparrow}, ?A$*. Then* $P \equiv \overline{x}\langle\vec{y}\rangle$ *for some* $\{\vec{y}\} \subseteq \mathsf{fn}(A)$*.*

Lemma 20. $P \mapsto\!\!\!\!\!\rightarrow Q \Rightarrow \mathsf{inn}(P) = \mathsf{inn}(Q)$*.*

The proofs of these 4 statements are straightforward. The next lemma is useful because it constraints the syntax of processes of translated extended System F type. This is handy in the definability proof. By *translated extended System F type* we mean types of the form $\alpha^{\circ}\{\vec{\mathrm{X}}^{\exists}_{\tau}/\vec{\mathrm{X}}\}$ or $\alpha^{\bullet}\{\vec{\mathrm{X}}^{\exists}_{\tau}/\vec{\mathrm{X}}\}$ for some System F type α.

Lemma 21. (syntactic shape) *Let* P *be typable by a translated extended System F type, maximally copycatted and* $P \not\mapsto$*. Then we have the following possibilities for typings that allow* P *to have an output as initial action.*

1. $P \equiv \overline{u}\langle x\rangle$ *and* $\vdash_{\mathsf{O}} P \triangleright u : (\mathrm{X})^{\uparrow}, x : \overline{\mathrm{X}}, \overline{E^{\circ}}$.
2. $P \equiv \overline{u}(x)!x(\vec{v}).Q$, $\vdash_{\mathsf{O}} P \triangleright u : (\mathrm{X}^{\exists}_{(\vec{\tau})^!})^{\uparrow}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright \vec{v} : \vec{\tau}, \overline{E^{\circ}}^{\,-x}.$$
3. $P \equiv \overline{u}(x)!x(v).Q$, $\vdash_{\mathsf{O}} P \triangleright u : (\forall \mathrm{X}^{\forall}.\beta)^{\bullet}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright v : \beta^{\bullet}, \overline{E^{\circ}},$$
where $\mathrm{X}^{\forall} \notin \mathsf{ftv}(E)$.
4. $P \equiv \overline{u}(x)!x(vr).Q$, $\vdash_{\mathsf{O}} P \triangleright u : (\alpha \Rightarrow \beta)^{\bullet}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright v : \overline{\alpha^{\circ}}, r : \beta^{\bullet}, \overline{E^{\circ}}.$$
5. $P \equiv \overline{x}(y)y(v).Q$, $\vdash_{\mathsf{O}} P \triangleright u : \alpha^{\bullet}, x : \overline{(\forall \mathrm{X}^{\forall}.\beta)^{\circ}}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright u : \alpha^{\bullet}, v : \overline{\beta^{\circ}}\{\mathrm{X}^{\exists}/\mathrm{X}^{\forall}\}, x : \overline{(\forall \mathrm{X}^{\forall}.\beta)^{\circ}}, \overline{E^{\circ}}$$
Here $\mathrm{X}^{\forall} \notin \mathsf{ftv}(E, \alpha)$.
6. $P \equiv \overline{x}(yr)(!y(v).Q \mid r(v).R)$ *and* $\vdash_{\mathsf{O}} P \triangleright u : \alpha^{\bullet}, x : \overline{((\forall \mathrm{X}^{\forall}.\beta) \Rightarrow \gamma)^{\circ}}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright v : \beta^{\bullet}, \overline{F^{\circ}}, \qquad \vdash_{\mathsf{O}} R \triangleright u : \alpha^{\bullet}, v : \overline{\gamma^{\circ}}, \overline{G^{\circ}},$$
where $\overline{F^{\circ}} \asymp \overline{G^{\circ}}$, $\overline{F^{\circ}} \odot \overline{G^{\circ}} = x : \overline{((\forall \mathrm{X}^{\forall}.\beta) \Rightarrow \gamma)^{\circ}}, \overline{E^{\circ}}$.
7. $P \equiv \overline{x}(yr)(!y(vw).Q \mid z(v).R)$ *and* $\vdash_{\mathsf{O}} P \triangleright u : \alpha^{\bullet}, x : \overline{((\beta \Rightarrow \gamma) \Rightarrow \delta)^{\circ}}, \overline{E^{\circ}}$ *and*
$$\vdash_{\mathsf{O}} Q \triangleright v : \overline{\beta^{\circ}}, w : \gamma^{\bullet}, \overline{F^{\circ}}, \qquad \vdash_{\mathsf{O}} R \triangleright u : \alpha^{\bullet}, v : \overline{\delta^{\circ}}, \overline{G^{\circ}},$$
where $\overline{F^{\circ}} \asymp \overline{G^{\circ}}$, $\overline{F^{\circ}} \odot \overline{G^{\circ}} = x : \overline{((\beta \Rightarrow \gamma) \Rightarrow \delta)^{\circ}}, \overline{E^{\circ}}$.
8. $P \equiv (\nu z)(\overline{x}\langle yz\rangle \mid z(v).Q)$ *and* $\vdash_{\mathsf{O}} P \triangleright u : \alpha^{\bullet}, x : \overline{(\mathrm{Y} \Rightarrow \beta)^{\circ}}, \overline{E^{\circ}}$, $E(y) = \mathrm{Y}$,
$$\vdash_{\mathsf{O}} Q \triangleright u : \alpha^{\bullet}, v : \overline{\beta^{\circ}}, \overline{E^{\circ}}.$$

9. $P \equiv \overline{x}(yr)(!y(\vec{v}).Q \mid r(v).R)$, $\vdash_0 P \rhd u : \alpha^\bullet, x : \overline{(\mathrm{Y}^\exists \Rightarrow \beta)^\circ}, \overline{E^\circ}$ *and*

$$\vdash_0 Q \rhd \vec{v} : \vec{\tau}, \overline{F^\circ}, \qquad \vdash_0 R \rhd u : \alpha^\bullet, v : \overline{\beta^\circ}, \overline{G^\circ}.$$

where $\overline{F^\circ} \asymp \overline{G^\circ}$, $\overline{F^\circ} \odot \overline{G^\circ} = x : \overline{(\mathrm{Y}^\exists \Rightarrow \beta)^\circ}, \overline{E^\circ}$.

Proof. By induction on the derivation of typing judgements. We start by considering the initial linear output at u and the carried name being typed with a type variable. Clearly the process given by (1) works. We show there be no others up to $\equiv$. Assume $\overline{u}\langle x\rangle|Q$, where $\vdash_{\mathtt{I}} Q \rhd F$ for some suitable F. As $\overline{u}\langle x\rangle|Q \not\rightarrow$, we can apply Lemma 6.3 and conclude that $Q \equiv 0$. It is also impossible to have $\overline{u}\langle x\rangle$ where x is typed by an existentially annotated type variable because otherwise P would not be maximally copycatted. That leaves $(\nu\vec{z})(\overline{u}\langle x\rangle|Q)$ where u is not restricted. If x is also free, we can immediately reduce to the previous case. Next assume the initial output at u is typed by the translation of a universal abstraction. Then $\overline{u}\langle x\rangle$ is impossible because we are dealing only with maximally copycatted processes. Hence we must have $(\nu\vec{z})(R \mid \overline{u}(x)!x(v).Q)$ where u is free. As in the previous case, we show that $R \equiv 0$, which leaves just (3). When the initial output at u is the translation of a function, we proceed similarly.

Now we assume the initial output at x is replicated. We cannot have $(\nu\vec{z})(\overline{x}\langle y\rangle \mid Q)$, with x and y being free, because P is maximally copycatted. Hence the process must be of the form $R \mid \overline{x}(y)Q$. As before, we show that $R \equiv 0$. Since we can only restrict y is there's matching input, our process must in fact be $\overline{x}(y)y(v).Q$, as stated by (5).

The remaining cases are similar. □

We need to manipulate innocent functions. That requires a bit of convenient notation. Let $\mathbf{\Psi}$ be an innocent function. Assuming typability, $\overline{x}(\vec{y})\mathbf{\Psi}$ is the innocent function obtained from $\mathbf{\Psi}$ by prefixing any $\mathbf{\Psi}$-trace with $\overline{x}(\vec{y})$ and then taking prefix-closure; $x(\vec{y})\mathbf{\Psi}$ and $!x(\vec{y}).\mathbf{\Psi}$ are defined similarly. Clearly $\overline{x}(\vec{y})\mathsf{inn}(P) = \mathsf{inn}(\overline{x}(\vec{y}).P)$ etc.

Lemma 22. *Let $\mathbf{\Psi}$ be finite innocent. Then $\sharp(\mathbf{\Psi}) < \sharp(\overline{x}(\vec{y})\mathbf{\Psi})$ assuming typability.*

Let σ be a function from existentially annotated type variables to types of the form $[n]_\lambda$ where $n > 0$. Assume $\mathbf{\Psi}$ is an innocent function. We say σ *offers enough choice for* $\mathbf{\Psi}$ if whenever a $\mathbf{\Psi}$-node $\vdash_{\mathtt{I}} P \rhd A$ can do k inputs

$$\vdash_{\mathtt{I}} P \rhd A \xrightarrow{x\langle(\nu\vec{a_i})\vec{b_i}\rangle} \cdots \qquad (i = 1, ..., k)$$

then for each b_{i_j} from $\vec{b_i}$ that is not in $\vec{a_i}$:

$$A(b_{i_j}) = \mathrm{X}^\exists_\tau \text{ implies } (\sigma(\mathrm{X}^\exists_\tau) = [m]_\lambda \Rightarrow k \le m).$$

A *concretiser for* $\mathbf{\Psi}$ is a mapping like σ above, provided it offers enough choice for $\mathbf{\Psi}$ and has a finite domain containing all of $\mathbf{\Psi}$'s free, existentially annotated type variables. Clearly, if $\boldsymbol{\Psi}$ is finite innocent, we can always find corresponding concretisers. As a matter of notation, we assume concretisers are postfix operators. We apply them to types, terms and type-environments alike, where they substitute all free existentially annotated type variables.

Proposition 25. (definability) *Let $\boldsymbol{\Psi}$ be a finite innocent function of translated extended System F type $u : \alpha^\bullet, \overline{E^\circ}$. Assume σ is a concretiser for $\boldsymbol{\Psi}$. Then there is a System F term $E \vdash M : \alpha$ such that $\mathsf{inn}(E\sigma \vdash M\sigma : \alpha\sigma) = \boldsymbol{\Psi}$.*

Proof. We proceed by induction on the size of $\boldsymbol{\Psi}$. Clearly $\sharp(\boldsymbol{\Psi}) = 0$ is impossible by Lemma 17. So the base case is $\sharp(\boldsymbol{\Psi}) = 1$.

Let $\vdash_{\mathsf{O}} P \rhd u : \alpha^\bullet, \overline{E^\circ}$ with $\mathsf{inn}(P) = \boldsymbol{\Psi}$ and $P \not\mapsto$. Restricting P to be without $\mapsto$-redexes is possible because each typable process has a $\mapsto$-normal form (Lemma 9) and processes equated by $\mapsto\!\!\!\!\twoheadrightarrow$ induce the same innocent functions (Lemma 20). In addition we assume P to be maximally copycatted. Lemmas 10 and 14 guarantee that this is always possible and does not affect innocent functions.

Now Lemma 18 together with $\sharp(\boldsymbol{\Psi}) = 1$ prevents any names being bound by the initial (and only) visible action. Clearly we can also not have an action $\vdash_{\mathsf{O}} P \rhd u : \alpha^\bullet, \overline{E^\circ} \xrightarrow{\overline{x}\langle y \rangle} \cdots$ where $x \in \mathsf{fn}(\overline{E^\circ})$. This is because otherwise we'd have to have $\overline{E^\circ}(x) = \overline{\beta^\circ} = (\mathrm{Y}^\forall)^p$, $p \in \{\uparrow, \boldsymbol{?}\}$. But this is impossible as a quick induction on the structure of β shows. Hence our unique visible action must be

$$\vdash_{\mathsf{O}} P \rhd u : \alpha^\bullet, \overline{E^\circ} \xrightarrow{\overline{u}\langle x \rangle} \vdash_{\mathtt{I}} Q \rhd \overline{E^\circ}.$$

where $\alpha = \mathrm{Y}$. By Lemma 19 this means $P \equiv \overline{u}\langle x \rangle$ and $\overline{E^\circ} = y : \mathrm{Y}^\forall, \overline{F^\circ}$. The corresponding System F term is simply $E\sigma \vdash x : \mathrm{Y}$, as $[\![E\sigma \vdash x : \mathrm{Y}]\!]_u = \overline{u}\langle x \rangle$. This concludes the base case.

The inductive step is more involved. We begin by distinguishing two cases: the initial visible action happens at u or it does not. Let's start with the former. By Lemma 21 we have 2 cases.

- $P \equiv \overline{u}\langle x \rangle, E(x) = \mathrm{X}^\forall$. This is in fact the base case.
- $P \equiv \overline{u}(c)Q$. Now we have 2 subcases.
  - $\alpha = (\beta \Rightarrow \gamma)$, i.e. $\alpha^\circ = (\overline{\beta^\circ}\gamma^\bullet)^!$ Then by Lemma 21: $P \equiv \overline{u}(c)!c(xm).Q$ where $\vdash_{\mathsf{O}} Q \rhd x : \overline{\beta^\circ}, m : \gamma^\bullet, \overline{E^\circ}$ and $\mathsf{inn}(Q)$ is strictly shorter than $\mathsf{inn}(P)$ (Lemma 22). By (IH) we can hence find M such that $\mathsf{inn}(Q) = \mathsf{inn}([\![E\sigma \vdash M\sigma : \alpha\sigma]\!]_m)$. Now

    $$\mathsf{inn}([\![\lambda x.M]\!]_u) = \mathsf{inn}(\overline{u}(c)!c(xm).Q) = \boldsymbol{\Psi}.$$

  - $\alpha = \forall \mathrm{X}.\beta$, i.e. $\forall \mathrm{X}.(\beta^\bullet)^!$. Using Lemma 21 again, we get $P \equiv \overline{u}(c)!c(m).Q$ and as in the previous case the (IH) yields a System F term M such that

    $$\mathsf{inn}([\![\Lambda \mathrm{X}.M]\!]_u) = \mathsf{inn}(\overline{u}(c)!c(m).Q) = \boldsymbol{\Psi}.$$

This exhausts initial outputs at u. We must now tackle the more complicated case of the initial output happening at a free name in $\overline{E^\circ}$. We have the following cases (Lemma 21).

– $\vdash_0 P \triangleright u : \alpha^\bullet, x : \overline{\forall \mathrm{X}.(\beta^\circ)}, \overline{E^\circ}$. Abbreviating the type of P to A, we can do the following transitions.

$$\vdash_0 P \triangleright A \xrightarrow{\overline{x}(a)} \xrightarrow{a(b)} \vdash_0 Q \triangleright A, b : \overline{\beta^\circ}\{\mathrm{X}^\exists_\tau/\mathrm{X}^\forall\}$$

By the (VC), $\mathrm{X}^\exists_\tau$ is not in the domain on σ. We must hence construct an appropriate σ' such that $\mathsf{ftv}^\exists(A, b : \overline{\beta^\circ}\{\mathrm{X}^\exists_\tau/\mathrm{X}^\forall\}) \subseteq \mathsf{dom}(\sigma')$. For this purpose, let $\mathbf{\Psi}_Q = \mathsf{inn}(Q)$. A $\mathbf{\Psi}_Q$-node $\vdash_\mathtt{I} R \triangleright B$ is a *free branch* if it has a transition of the form $\vdash_\mathtt{I} R \triangleright B \xrightarrow{a\langle b\rangle} \cdots$ or $\vdash_\mathtt{I} R \triangleright B \xrightarrow{a\langle(\nu b)bc\rangle} \cdots$. Using Subject Transition (Proposition 12) together with that fact that P is of translated extended System F type, one sees immediately that B is also made from translated extended System F types. Hence all free inputs in $\mathbf{\Psi}_Q$ must be by free branches. Call a free branch R *initial* if there are no other free branches on the path along the transitions from Q to R. Let the initial free branches be $R_1, ..., R_m$ for some $m \geq 0$. It is possible for m to be 0 or greater than 1. Each R_i is typed $\vdash_\mathtt{I} R_i \triangleright A_i$. By the (VC) we can assume that if $A_i(c) = \mathrm{X}^\exists$ and $A_j(c) = \mathrm{Y}^\exists$ then $\mathrm{X} = \mathrm{Y}$. Let

$$\{b_i^1, ..., b_i^{k_i}\} = \{c \mid \exists \mathrm{X}.A_i(c)\mathrm{X}^\exists\}$$

Clearly k_i is always greater than 0. Otherwise R_i would not be a free branch. Set $n = k_1 + \cdots + k_m$. Now define

$$\sigma'(\mathrm{Y}^\exists_\gamma) = \begin{cases} \sigma(\mathrm{Y}^\exists_\gamma) & \mathrm{X} \neq \mathrm{Y}, \\ [n]_\lambda & \mathrm{X} = \mathrm{Y}. \end{cases}$$

By its construction, σ' has sufficient choice for $\mathbf{\Psi}_Q$. Hence we can apply the (IH) to get a term $(E, x : \forall \mathrm{X}\beta, b : \beta)\sigma' \vdash M\sigma' : \alpha\sigma'$ such that

$$\mathsf{inn}((E, x : \forall \mathrm{X}\beta, b : \beta)\sigma' \vdash M\sigma' : \alpha\sigma') = \mathbf{\Psi}_Q.$$

Noting now that $[\![(\lambda b.M)(x[n]_\lambda)]\!]_u \mapsto\!\!\!\!\to \overline{x}(a)a(b).[\![M]\!]_u$ and using Lemma 20 we get

$$\begin{aligned} \mathsf{inn}([\![(\lambda b.M)(x[n]_\lambda)]\!]_u) &= \mathsf{inn}(\overline{x}(a)a(b).[\![M]\!]_u) \\ &= \overline{x}(a)a(b).\mathsf{inn}([\![M]\!]_u) \\ &= \overline{x}(a)a(b).\mathbf{\Psi}_q \\ &= \mathbf{\Psi} \end{aligned}$$

as required.

– $P \equiv \overline{x}(yr)(!y(v).Q \mid r(v).R)$ and $\vdash_0 P \triangleright u : \alpha^\bullet, x : \overline{((\forall \mathrm{X}^\forall.\beta) \Rightarrow \gamma)^\circ}, \overline{E^\circ}$ where $\vdash_0 Q \triangleright v : \beta^\bullet, \overline{F^\circ}$, $\vdash_0 R \triangleright u : \alpha^\bullet, v : \overline{\gamma^\circ}, \overline{G^\circ}$. Clearly, Q and R have finite innocent functions smaller that $\mathbf{\Psi}$, so by (IH), we can find System F terms $F\sigma \vdash M\sigma : \beta\sigma$ and $G\sigma, v : \gamma\sigma \vdash N\sigma : \alpha\sigma$ such that (omitting types for readability) $\mathsf{inn}(Q) = \mathsf{inn}([\![M]\!]_v)$ and $\mathsf{inn}(R) = \mathsf{inn}([\![N]\!]_u)$. Then clearly

$$\mathbf{\Psi} = \mathsf{inn}([\![(\lambda v.N)(x(\Lambda \mathrm{X}.M))]\!]_u).$$

– $P \equiv \overline{x}(yr)(!y(vw).Q \mid z(v).R)$ and $\vdash_{\mathsf{0}} P \rhd u : \alpha^{\bullet}, x : \overline{((\beta \Rightarrow \gamma) \Rightarrow \delta)^{\circ}}, \overline{E^{\circ}}$ and $\vdash_{\mathsf{0}} Q \rhd v : \overline{\beta^{\circ}}, w : \gamma^{\bullet}, \overline{F^{\circ}}$, $\vdash_{\mathsf{0}} R \rhd u : \alpha^{\bullet}, v : \overline{\delta^{\circ}}, \overline{G^{\circ}}$. By (IH) we find System F terms $F\sigma, v : \beta\sigma \vdash M\sigma : \gamma\sigma$ and $G\sigma, v : \delta\sigma \vdash N\sigma : \alpha\sigma$ such that $\mathsf{inn}(Q) = \mathsf{inn}([\![M]\!]_v)$ and $\mathsf{inn}(R) = \mathsf{inn}([\![N]\!]_u)$. Then

$$\mathbf{\Psi} = \mathsf{inn}([\![(\lambda v.N)(x(\lambda v.M))]\!]_u).$$

– $P \equiv (\nu z)(\overline{x}\langle yz \rangle \mid z(v).Q)$ and $\vdash_{\mathsf{0}} P \rhd u : \alpha^{\bullet}, x : \overline{((\mathrm{Y} \Rightarrow \beta)^{\circ}}, \overline{E^{\circ}}$, $E(y) = \mathrm{Y}$, $\vdash_{\mathsf{0}} Q \rhd u : \alpha^{\bullet}, v : \overline{\beta^{\circ}}, \overline{E^{\circ}}$. By (IH) there's $E\sigma, v : \beta\sigma \vdash M\sigma : \alpha\sigma$ such that $\mathsf{inn}(Q) = \mathsf{inn}([\![M]\!]_v)$. Thus clearly

$$\mathbf{\Psi} = \mathsf{inn}([\![(\lambda v.M)(xy)]\!]_u).$$

– $P \equiv \overline{x}(yr)(!y(\vec{v}).Q \mid r(v).R)$, $\vdash_{\mathsf{0}} P \rhd u : \alpha^{\bullet}, x : \overline{(\mathrm{Y}^{\exists}_{\gamma} \Rightarrow \beta)^{\circ}}, \overline{E^{\circ}}$ and $\vdash_{\mathsf{0}} Q \rhd \vec{v} : \vec{\tau}, \overline{F^{\circ}}$, $\vdash_{\mathsf{0}} R \rhd u : \alpha^{\bullet}, v : \overline{\beta^{\circ}}, \overline{G^{\circ}}$. Let A be the above type for P and assume that $\sigma(\mathrm{Y}^{\exists}_{\gamma}) = [m]_{\lambda}$. We have 2 subcases.
  - $\beta = \mathrm{X}^{\exists}_{\delta}$ and $\sigma(\mathrm{X}^{\exists}_{\delta}) = [n]_{\lambda}$. In this case we have the following transition.

$$P \xrightarrow{\overline{x}(yr)} \vdash_{\mathrm{I}} !y(\vec{v}).Q \mid r(v).R \rhd y : \mathrm{Y}^{\exists}_{\gamma}, r : (\mathrm{X}^{\exists}_{\delta})^{\downarrow}$$

Now let $k \leq n$ be the number distinct entries $z_i : \mathrm{X}^{\exists}_{\delta}$ in $\overline{E^{\circ}}$. Then there are k distinct transitions

$$\vdash_{\mathrm{I}} !y(\vec{v}).Q \mid r(v).R \rhd y : \mathrm{Y}^{\exists}_{\gamma}, r : (\mathrm{X}^{\exists}_{\delta})^{\downarrow} \xrightarrow{r\langle z_i \rangle} \vdash_{\mathsf{0}} Q_i \rhd A$$

By (IH) we get corresponding $M_1, ..., M_k$ such that

$$\mathsf{inn}([\![E, x : [m]_{\lambda} \Rightarrow [n]_{\lambda} \vdash M_i : \alpha\sigma]\!]) = \mathsf{inn}(Q_i)$$

Now define

$$N_i \stackrel{def}{=} \begin{cases} M_i & 1 \leq i \leq k \\ M_1 & k < i \leq n. \end{cases}$$

We can do this because σ has enough choice, so $k \leq n$. That $k > 0$ is because we are in the inductive step: so by the inductive assumption $\mathbf{\Psi}$'s cardinality exceeds 1. But then P must be able to do at least two consecutive visible transitions, which would not be possible if $k = 0$. We are almost there. The target term, corresponding to $\mathbf{\Psi}$ is $(x\ M\ \alpha\ N_1...N_n)\sigma$, where M is typable as $E\sigma \vdash M\sigma : [n]_{\lambda}$. Now we consider what transitions the translation of M has in $\mathbf{\Psi}$: There are two possibilities: either there are no such transitions, i.e. the environment working at x ignores the translation of M. Then we set $M \stackrel{def}{=} \pi^m_1$. Alternatively, the environment invokes the translation of M. By innocence, all invocations give the same result, say choosing the i-th possibility out of n. Then we set $M \stackrel{def}{=} \pi^m_i$. Then

$$\mathsf{inn}([\![E\sigma, x : [m]_{\lambda} \Rightarrow [n]_{\lambda} \vdash (x\ M\ \alpha\ N_1...N_n)\sigma : \alpha\sigma]\!]_u) = \mathbf{\Psi}.$$

- β is not an existentially annotated type variable. This case is essentially like the last, except that $n = 1$.

$\square$

Finally we note the following. Below we say P is *finite* if its innocent function is finite.

Lemma 23. *Let* $\vdash_{\mathsf{O}} P_{1,2} \triangleright y : \tau^{\uparrow}$ *and* $\vdash_{\mathtt{I}} R \triangleright y : \overline{\tau}, x : \mathbb{B}$. *Then* $P_1|R \Downarrow_{\mathsf{t}\langle x\rangle}$ *and* $P_2|R \Downarrow_{\mathsf{f}\langle x\rangle}$ *imply for some finite* $\vdash_{\mathtt{I}} R' \triangleright y : \overline{\tau}, x : \mathbb{B}$ *we have* $P_1|R' \Downarrow_{\mathsf{t}\langle x\rangle}$ *and* $P_2|R' \Downarrow_{\mathsf{t}\langle x\rangle}$.

Proof. (outline) We take $\mathsf{inn}(R)$, and mark just those parts which are used to derive $P_1|R \Downarrow_{\mathsf{t}\langle x\rangle}$ and $P_2|R \Downarrow_{\mathsf{f}\langle x\rangle}$. These parts are obviously finite. Unmarked parts can be infinite, but they do not concern the extended reductions involved in the above convergences. Thus it suffices to replace each of these unmarked parts with a finite behaviour of the same type, since, if we can do so, we can reconstruct a process defining this finite behaviour in the same way we did in the definability lemma. For replacing each such part, it suffices to show there is always at least one finite process for each well-formed action type. Such a process is easily constructed by induction on (extended) action types, following the term constructor corresponding to positive types ($\uparrow$ and $\mathbf{!}$) and a linear input as needed, without using $\mathbf{?}$-actions (note the resulting behaviour is what corresponds to a constant function, which is immediately finite). $\square$

Theorem 4. (full abstraction) *Let* $\vdash M : \alpha$. *Then* $M \cong_{\forall} N$ *iff* $[\![M]\!]_u \cong_{\forall\exists} [\![N]\!]_u$.

Proof. In the light of Corollary 1 we need only show completeness. The argument is standard [11, 48]. Assume $[\![M]\!]_u \not\cong_{\forall\exists} [\![N]\!]_u$. Then we can find a finite R such that $(\nu \vec{x}u)([\![M]\!]_u \mid R) \Downarrow_e [\![\mathsf{T}]\!]_y$ but $(\nu \vec{x}u)([\![M]\!]_u \mid R) \Downarrow_e [\![\mathsf{F}]\!]_y$ By Lemma 23 it suffices to consider finite Rs. By definability, we can find L such that $[\![L]\!]_y \cong_{\forall\exists} R$. Hence $(\nu \vec{x}u)([\![M]\!]_u \mid [\![L]\!]_y) \not\cong_{\forall\exists} (\nu \vec{x}u)([\![N]\!]_u \mid [\![L]\!]_y)$, a contradiction. $\square$

7.3 Call-by-Name and Type Isomorphisms

The encoding of System F we have explored was CBV. We could have used CBN instead. It may be interesting and illustrative of the power of our syntax-based approach to modelling programming languages to see how CBV's full abstraction leads to an *easy* proof of full abstraction for CBN, just by looking at the types of the two translations.

Below $\mathsf{C}[\cdot]^{B,\psi}_{A,\phi}$ indicates a typed context where the hole has type A, ϕ and the result B, ψ. A, ϕ and B, ψ are $\cong_{\forall\exists}$-*isomorphic* if for some $\mathsf{C}_1[\cdot]^{B,\psi}_{A,\phi}$ and $\mathsf{C}_2[\cdot]^{A,\phi}_{B,\psi}$, we have $P \cong_{\forall\exists} \mathsf{C}_2[\mathsf{C}_1[P]^{B,\psi}_{A,\phi}]^{A,\phi}_{B,\psi}$ for each $\vdash_{\phi} P \triangleright A$ and vice versa. In particular we write $\tau_1 \cong_{\forall\exists} \tau_2$ if $x_1 : \tau_1$ and $x_2 : \tau_2$ are $\cong_{\forall\exists}$-isomorphic (the omission of names loses no precision since the isomorphism does not depend on the particular choice of names). Contexts can often be defined by parallel composition with hiding.

Proposition 26. *With* $\mathsf{md}(\tau) = \mathbf{!}$ *we have (1)* $\tau \cong_{\forall\exists} (\tau)^{\uparrow}$ *and (2)* $(\tau)^{\uparrow} \cong_{\forall\exists} ((\tau)^{\uparrow})^{\mathbf{!}}$.

Proof. The contexts for (1) are

$$C[\cdot]^{y:(\tau)^{\uparrow}}_{x:\tau} \stackrel{def}{=} \overline{y}(x)[\cdot] \qquad D[\cdot]^{x:\tau}_{y:(\tau)^{\uparrow}} \stackrel{def}{=} !x(\vec{v}).(\nu y)([\cdot] \mid y(w).\overline{w}\langle\vec{v}\rangle)$$

while for (2) we use

$$C[\cdot]^{y:((\tau)^{\uparrow})^{!}}_{x:(\tau)^{\uparrow}} \stackrel{def}{=} !y(x).[\cdot] \qquad D[\cdot]^{x:(\tau)^{\uparrow}}_{y:((\tau)^{\uparrow})^{!}} \stackrel{def}{=} \overline{x}(a)(!a(v)\overline{y}(b)[b \to v]^{\tau} \mid [\cdot]).$$

It is straightforward to check that these context do indeed witness the isomorphisms stated in (1) and (2). □

Now we give the CBN encoding and prove full abstraction.

$$\begin{array}{l}
\alpha^{\star} \stackrel{def}{=} (\alpha^{\diamond})^{!} \quad \mathrm{X}^{\diamond} \stackrel{def}{=} \mathrm{X}^{\uparrow} \quad (\alpha \Rightarrow \beta)^{\diamond} \stackrel{def}{=} (\overline{\alpha^{\diamond}}\beta^{\star})^{!} \quad (\forall \mathrm{X}.\alpha)^{\diamond} \stackrel{def}{=} \forall \mathrm{X}.(\alpha^{\star})^{!} \\
\langle\!\langle x \rangle\!\rangle_u \stackrel{def}{=} [u \to x]^{\alpha^{\star}} \qquad \langle\!\langle \lambda x^{\alpha}.M \rangle\!\rangle_u \stackrel{def}{=} !u(xz).\overline{z}(m)\langle\!\langle M \rangle\!\rangle_z \\
\langle\!\langle MN \rangle\!\rangle_u \stackrel{def}{=} !u(xy).(\boldsymbol{\nu}\, m)(\langle\!\langle M \rangle\!\rangle_m \mid \overline{m}(nr)(\langle\!\langle N \rangle\!\rangle_n \mid r(w).\overline{w}\langle xy \rangle)) \\
\langle\!\langle \Lambda \mathrm{X}.M \rangle\!\rangle_u \stackrel{def}{=} !u(a).\overline{a}(m)\langle\!\langle M \rangle\!\rangle_m \qquad \langle\!\langle M\beta \rangle\!\rangle_u \stackrel{def}{=} !u(r).(\nu m)(\langle\!\langle M \rangle\!\rangle_m \mid \overline{m}\langle r \rangle)
\end{array}$$

Lemma 24. *It is the case that $\alpha^{\star} \cong_{\forall\exists} \alpha^{\bullet}$ for all System F types α.*

Proof. Straightforward from Proposition 26. □

Theorem 5. (full abstraction for CBN) $\vdash M = N : \alpha$ *iff* $\vdash_{\mathtt{I}} \langle\!\langle M \rangle\!\rangle_u \cong_{\forall\exists} \langle\!\langle N \rangle\!\rangle_u \triangleright u : \alpha^{\star}$.

Proof. Lemma 24 leaves no doubt that $[\![M]\!]_u \cong_{\forall\exists} [\![N]\!]_u$ iff $\langle\!\langle M \rangle\!\rangle_u \cong_{\forall\exists} \langle\!\langle N \rangle\!\rangle_u$, so the result is a trivial consequence of CBV's full abstraction. □

References

1. Abadi, M., Cardelli, L., and Curien, P.-L. Formal parametric polymorphism. *TCS 121*, 1-2 (1993), 9–58.
2. Abramsky, S. Computational interpretations of linear logic. *TCS 111* (1993).
3. Abramsky, S., and Jagadeesan, R. A game semantics for generic polymorphism. In *Proc. FOSSACS'03* (April 2003), no. 2620 in LNCS, Springer, pp. 1–22.
4. Abramsky, S., Jagadeesan, R., and Malacaria, P. Full abstraction for PCF. *Information and Computation 163* (2000), 409–470.
5. Abramsky, S., and Lenisa, M. Axiomatizing fully complete models for ML polymorphic types. In *Proc. of MFCS'2000* (2000).
6. Abramsky, S., and Lenisa, M. A fully-complete PER model for ML polymorphic types. In *Proc. of CSL'2000*, LNCS. Springer, 2000.
7. Barendregt, H. *The Lambda Calculus*. North Holland, 1985.
8. Barendregt, H. Lambda calculi with types. In *Handbook of Logic in Computer Science. Background: Computational Structures*, S. Abramsky, D. Gabbay, and T. Maibaum, Eds., vol. 2. Clarendon Press, 1992.
9. Berger, M. *Towards Abstractions for Distributed Systems*. PhD thesis, Imperial College, London, 2002.

10. Berger, M., Honda, K., and Yoshida, N. Sequentiality and the π-calculus. Full version of [11].
11. Berger, M., Honda, K., and Yoshida, N. Sequentiality and the π-calculus. In *Proc. TLCA'01* (2001).
12. Berger, M., Honda, K., and Yoshida, N. Genericity and the π-calculus. In *Proc. FOSSACS'03* (April 2003), no. 2620 in LNCS, Springer, pp. 103–119.
13. Boudol, G. Asynchrony and the pi-calculus. Tech. Rep. 1702, INRIA, 1992.
14. Bracha, G., Odersky, M., Stoutamire, D., and Wadler, P. Making the future safe for the past: Adding genericity to the Java programming language. In *Proceedings of OOPSLA 98* (October 1998).
15. Girard, J.-Y. *Interprétation Fonctionnelle et Élimination des Coupures de l'Arithmétique d'Ordre Supérieur.* PhD thesis, Universite de Paris VII, 1972.
16. Girard, J.-Y. Linear logic. *TCS 50* (1987).
17. Grossman, D. Existential types for imperative languages. In *ESOP02* (2002), LNCS 2305, Springer, pp. 21–35.
18. Honda, K., and Tokoro, M. An object calculus for asynchronous communication. In *Proceedings of ECOOP'91* (1991).
19. Honda, K., and Yoshida, N. Game-theoretic analysis of call-by-value computation (extended abstract). In *Proc. of ICALP'97* (1997), LNCS, Springer Verlag.
20. Honda, K., and Yoshida, N. Game-theoretic analysis of call-by-value computation. *TCS 221* (1999), 393–456.
21. Honda, K., and Yoshida, N. A uniform type structure for secure information flow. In *POPL'02* (2002), pp. 81–92.
22. Honda, K., Yoshida, N., and Berger, M. Control in the π-calculus. In *Proc. CW'04* (2004), ACM Press.
23. Hughes, D. J. D. Games and definability for system F. In *LICS'97* (1997), IEEE Computer Society Press, pp. 76–86.
24. Hyland, J. M. E., and Ong, C. H. L. On full abstraction for PCF. *Information and Computation 163* (2000), 285–408.
25. Kobayashi, N., Pierce, B., and Turner, D. Linear types and π-calculus. In *Proceedings of POPL'96* (1996), pp. 358–371.
26. Lazic, R., Newcomb, T., and Roscoe, A. On model checking data-independent systems with arrays without reset. Tech. Rep. RR-02-02, Oxford University, 2001.
27. Leroy, X. *Polymorphic typing of an algorithmic language.* PhD thesis, University of Paris, 1992.
28. Milner, R. Functions as processes. *MSCS 2*, 2 (1992), 119–141.
29. Milner, R. The polyadic π-calculus: A tutorial. In *Proceedings of the International Summer School on Logic Algebra of Specification* (1992), Marktoberdorf.
30. Milner, R. Speech on receiving an Honorary Degree from the University of Bologna, ICALP'97, 1997.
31. Milner, R., Parrow, J., and Walker, D. A calculus of mobile processes, parts I and II. *Info. & Comp. 100*, 1 (1992).
32. Milner, R., Tofte, M., and Harper, R. W. *The Definition of Standard ML.* MIT Press, 1990.
33. Mitchell, J. C. On the equivalence of data representation. In *Artificial Intelligence and Mathematical Theory of Computation* (1991).
34. Mitchell, J. C. *Foundations for Programming Languages.* MIT Press, 1996.
35. Murawski, A., and Ong, C.-H. L. Evolving games and essential nets for affine polymorphism. In *Proc. of TLCA'01* (2001), no. 2044 in LNCS, Springer.
36. Pierce, B., and Sangiorgi, D. Typing and subtyping for mobile processes. *MSCS 6*, 5 (1996), 409–454.

37. PIERCE, B., AND SANGIORGI, D. Behavioral equivalence in the polymorphic pi-calculus. *Journal of ACM 47*, 3 (2000), 531–584.
38. PIERCE, B. C., AND TURNER, D. N. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling, and M. Tofte, Eds. MIT Press, 2000.
39. PITTS, A., AND STARK, I. Operational reasoning for functions with local state. In *HOOTS'98* (1998), CUP, pp. 227–273.
40. PITTS, A. M. Existential Types: Logical Relations and Operational Equivalence. In *Proceedings ICALP'98* (1998), no. 1443 in LNCS, Springer, pp. 309–326.
41. PITTS, A. M. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science 10* (2000), 321–359.
42. PLOTKIN, G., AND ABADI, M. A logic for parameteric polymorphism. In *LICS'98* (1998), IEEE Press, pp. 42–53.
43. REYNOLDS, J. C. Types, abstraction and parametric polymorphism. In *Information Processing 83* (1983), R. E. A. Mason, Ed.
44. TOFTE, M. Type inference for polymorphic references. LNCS 2305, Springer, pp. 21–35.
45. TURNER, D. N. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1996.
46. VASCONCELOS, V. Typed concurrent objects. In *Proc. ECOOP'94* (1994).
47. VASCONCELOS, V. T., AND HONDA, K. Principal typing scheme for polyadic π-calculus. In *Proc. CONCUR'93* (1993).
48. YOSHIDA, N., BERGER, M., AND HONDA, K. Strong Normalisation in the π-Calculus. In *Proc. LICS'01* (2001), IEEE, pp. 311–322. The full version to appear in *Journal of Information and Computation*.
49. YOSHIDA, N., HONDA, K., AND BERGER, M. Linearity and bisimulation. In *FoSSaCs02* (2002), vol. 2303 of *LNCS*, Springer, pp. 417–433.

A Action Types

We write $x \to y$ if $x : \tau \to y : \tau'$ for some τ and τ', in a given action type. If x occurs in A and for no y we have $y \to x$ then we say x is *active in* A. $|A|$ denotes the set of nodes in A. We often write $x : \tau \in A$ instead of $x : \tau \in |A|$. We then write $A \asymp B$ iff:

- whenever $x{:}\tau \in A$ and $x{:}\tau' \in B$, $\tau \odot \tau'$ is defined; and
- whenever $x_1 \to x_2$, $x_2 \to x_3$, $\ldots$, $x_{n-1} \to x_n$ alternately in A and B ($n \geq 2$), we have $x_1 \neq x_n$.

Then $A \odot B$, defined iff $A \asymp B$, is the following action type.

- $x : \tau \in |A \odot B|$ iff either (1) $x \in (\mathsf{fn}(A) \setminus \mathsf{fn}(B)) \cup (\mathsf{fn}(B) \setminus \mathsf{fn}(A))$ and $x : \tau$ occurs in A or B; or (2) $x{:}\tau' \in A$ and $x{:}\tau'' \in B$ and $\tau = \tau' \odot \tau''$.
- $x \to y$ in $A \odot B$ iff $x{:}\tau_{\mathrm{I}}, y{:}\tau_{\mathrm{O}} \in |A \odot B|$ and $x = z_1 \to z_2$, $z_2 \to z_3, \ldots, z_{n-1} \to z_n = y$ ($n \geq 2$) alternately in A and B.

B Proof of Subject Reduction

We first define a permutability of rules, which is crucial in many results that follow. Let R and S be two rules which have one premises. We write $R \searrow S$ if, whenever there exists a derivation (A) in the right-hand side, then we have another deviation (B) in the left-hand side.

$$\text{(A)}\quad \dfrac{\dfrac{\vdash_\phi P \triangleright A}{\vdash_{\phi_1} P_1 \triangleright A_1}\, R}{\vdash_{\phi_2} P_2 \triangleright A_2}\, S \qquad\qquad \text{(B)}\quad \dfrac{\dfrac{\vdash_\phi P \triangleright A}{\vdash_{\phi_3} P_3 \triangleright A_3}\, S}{\vdash_{\phi_2} P_2 \triangleright A_2}\, R$$

If R has two premises and S has one we write $R \searrow_l S$ if, whenever there exists a derivation (A) in the right-hand side, then we have another deviation (B) in the left-hand side.

$$\text{(A)}\quad \dfrac{\dfrac{\vdash_\phi P \triangleright A \qquad \vdash_{\phi_1} P_1 \triangleright A_1}{\vdash_{\phi_2} P_2 \triangleright A_2}\, R}{\vdash_{\phi_3} P_3 \triangleright A_3}\, S \qquad\qquad \text{(B)}\quad \dfrac{\dfrac{\vdash_\phi P \triangleright A}{\vdash_{\phi_4} P_4 \triangleright A_4}\, S \qquad \vdash_{\phi_1} P_1 \triangleright A_1}{\vdash_{\phi_3} P_3 \triangleright A_3}\, R$$

The predicate $R \searrow_r S$ is similarly defined.

Lemma 25. *1. For any rule R with one premise other than* ($\textsc{In}^{!}$) *and* ($\textsc{In}^{\downarrow}$)*, we have $R \searrow$* ($\textsc{Weak}$).

2. ($\textsc{Weak}$) $\searrow$ ($\textsc{In}^{\downarrow}$) *if* ($\textsc{Weak}$) *does not weaken bound name y_i in $!x(\vec{y}).P$ of the conclusion of* ($\textsc{In}^{\downarrow}$)*. Similarly for* ($\textsc{In}^{!}$)*.*

3. Let $\psi \in \{l, r\}$, then ($\textsc{Weak}$) $\searrow_\psi$ ($\textsc{Par}$), ($\textsc{Par}$) $\searrow_\psi$ ($\textsc{Weak}$)

Proof. All straightforward from the typing rule schemes. □

Proposition 27. *If $\vdash_\phi P \triangleright A$ and $P \equiv Q$, then $\vdash_\phi Q \triangleright A$.*

Proof. By induction on the derivation of $P \equiv Q$.

$\underline{P|0 \equiv P}$**:** Now we use induction on the derivation of $\vdash_\phi P \triangleright A$. There are two cases: if the last rule used was ($\textsc{Weak}$), we use the inner induction hypothesis (IH). Otherwise we have a derivation

$$\dfrac{\vdash_\phi P \triangleright A \qquad \dfrac{\dfrac{\dfrac{}{\vdash_{\mathtt{I}} 0 \triangleright \emptyset}\,(\textsc{Zero})}{\vdots}\,(\textsc{Weak})}{\vdash_{\mathtt{I}} 0 \triangleright B}\,(\textsc{Weak}) \qquad A \asymp B}{\vdash_\phi P|0 \triangleright A \odot B}\,(\textsc{Par})$$

If the number of applications of ($\textsc{Weak}$) in this derivation is 0, the result is immediate, otherwise we use ($\textsc{Weak}$) $\searrow_r$ ($\textsc{Par}$). To push all weakenings below ($\textsc{Par}$) in the proof tree and reduce the problem to one already dealt with in the outermost first case. The converse direction is straightforward.

$\underline{(\nu x)(P|Q) \equiv P|(\nu x)Q}$**, if** $x \notin \mathsf{fn}(P)$**:** We proceed by induction on the derivation of the relevant typing judgement. Using the inner (IH) as above and (WEAK) $\searrow$ (RES), we can see that there are two interesting cases.

$$\dfrac{\dfrac{\vdash_\phi P \triangleright A \qquad \vdash_\psi Q \triangleright B \qquad \ldots}{\vdash_{\phi\odot\psi} P|Q \triangleright A \odot B = C, x : \tau}\text{(PAR)} \qquad \mathsf{md}(\tau) \in \{\bot, \mathbf{!}\}}{\vdash_{\phi\odot\psi} (\nu x)(P|Q) \triangleright C}\text{(RES)}$$

and

$$\dfrac{\dfrac{\dfrac{\vdash_\phi P \triangleright A \qquad \vdash_\psi Q \triangleright B \qquad \ldots}{\vdash_{\phi\odot\psi} P|Q \triangleright A^{-x} \odot B^{-x}}\text{(PAR)} \qquad \mathsf{md}(\tau) \in \{\bot, \mathbf{!}\}}{\vdash_{\phi\odot\psi} P|Q \triangleright C, x : \tau}\text{(WEAK)}}{\vdash_{\phi\odot\psi} (\nu x)(P|Q) \triangleright C}\text{(RES)}$$

In the former case we know from the assumptions that $x \notin \mathsf{fn}(A)$, $B = B', x : \tau$ and $C = A \odot B'$. Hence we derive as follows.

$$\dfrac{\vdash_\phi P \triangleright A \qquad \dfrac{\vdash_\psi Q \triangleright B', x : \tau}{\vdash_\psi (\nu x)Q \triangleright B'}\text{(RES)}}{\vdash_{\phi\odot\psi} P|(\nu x)Q \triangleright A \odot B' = C}\text{(PAR)}$$

To reduce the latter case to the former, we use (PAR) $\searrow_r$ (WEAK).
For the reverse direction, we also induce on the derivation of the typing judgement. Using the inner (IH) and permutations of weakening with parallel composition as above, we find essentially one interesting case.

$$\dfrac{\vdash_\phi P \triangleright A \qquad \dfrac{\vdash_\psi Q \triangleright B, x : \tau \qquad \mathsf{md}(\tau) \in \{\bot, \mathbf{!}\}}{\vdash_\psi (\nu x)Q \triangleright B}\text{(RES)}}{\vdash_{\phi\odot\psi} P|(\nu x)Q \triangleright A \odot B}\text{(PAR)}$$

By the (VC) we can assume $x \notin \mathsf{fn}(A)$. Hence we can construct:

$$\dfrac{\dfrac{\vdash_\phi P \triangleright A \qquad \vdash_\psi Q \triangleright B, x : \tau}{\vdash_{\phi\odot\psi} P|Q \triangleright A \odot (B, x : \tau) = (A \odot B), x : \tau}\text{(PAR)} \qquad \mathsf{md}(\tau) \in \{\bot, \mathbf{!}\}}{\vdash_{\phi\odot\psi} (\nu x)(P|Q) \triangleright A \odot B}\text{(RES)}$$

as required.

<u>All other cases</u>: Similar, but simpler.

□

Lemma 26. *1.* $\overline{\tau}\{\vec{\gamma}/\vec{\mathrm{x}}\} = \overline{\tau\{\vec{\gamma}/\vec{\mathrm{x}}\}}$.

2. If $\tau_1 \odot \tau_2$ *is defined, then so is* $\tau_1\{\vec{\gamma}/\vec{\mathrm{x}}\} \odot \tau_2\{\vec{\gamma}/\vec{\mathrm{x}}\}$.

3. If $A \asymp B$ *then also* $A\{\vec{\gamma}/\vec{\mathrm{x}}\} \asymp B\{\vec{\gamma}/\vec{\mathrm{x}}\}$ *and* $(A \odot B)\{\vec{\gamma}/\vec{\mathrm{x}}\} = (A\{\vec{\gamma}/\vec{\mathrm{x}}\}) \odot (B\{\vec{\gamma}/\vec{\mathrm{x}}\})$.

4. If $\vec{\mathrm{Y}} \cap \vec{\mathrm{Y}} = \emptyset$ *and* $\vec{\mathrm{X}} \cap \mathsf{ftv}(\vec{\delta}) = \emptyset$*, then* $\{\vec{\gamma}/\vec{\mathrm{X}}\}\{\vec{\delta}/\vec{\mathrm{Y}}\} = \{\vec{\delta}/\vec{\mathrm{Y}}\}\{\vec{\gamma}\{\vec{\delta}/\vec{\mathrm{Y}}\}/\vec{\mathrm{X}}\}$.

Proof. Straightforward. □

Lemma 27. *If* $\vdash_\phi P \triangleright A$ *then* $\vdash_\phi P \triangleright A\{\vec{\gamma}/\vec{\mathrm{X}}\}$.

Proof. By induction on the derivation of $\vdash_\phi P \triangleright A$.

(OUT): Assuming $\mathrm{Z}_i \notin \mathsf{ftv}(\vec{\tau}\{\vec{\xi}/\vec{\mathrm{Z}}\})$, we can not only infer

$$\vdash_{\mathsf{0}} \overline{x}\langle\vec{y}\rangle \triangleright x : \exists\vec{\mathrm{Z}}.(\vec{\tau})^{p_{\mathsf{0}}}, \vec{y} : \overline{\vec{\tau}\{\vec{\xi}/\vec{\mathrm{Z}}\}},$$

but also

$$\vdash_{\mathsf{0}} \overline{x}\langle\vec{y}\rangle \triangleright x : \exists\vec{\mathrm{Z}}.(\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\})^{p_{\mathsf{0}}}, \vec{y} : \overline{\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\}\{\vec{\xi}\{\vec{\gamma}/\vec{\mathrm{X}}\}/\vec{\mathrm{Z}}\}},$$

where $\mathrm{X}_i \notin \mathsf{ftv}(\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\})$ and $\{\vec{\mathrm{Z}}\} \cap \{\mathsf{ftv}(\vec{\gamma}), \vec{\mathrm{Y}}\} = \emptyset$. But $\exists\vec{\mathrm{Z}}.(\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\})^{p_{\mathsf{0}}} = (\exists\vec{\mathrm{Z}}.(\vec{\tau})^{p_{\mathsf{0}}})\{\vec{\gamma}/\vec{\mathrm{X}}\}$. Furthermore

$$\overline{\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\}\{\vec{\xi}\{\vec{\gamma}/\vec{\mathrm{X}}\}/\vec{\mathrm{Z}}\}} = \overline{\vec{\tau}}\{\vec{\gamma}/\vec{\mathrm{X}}\}\{\vec{\xi}\{\vec{\gamma}/\vec{\mathrm{X}}\}/\vec{\mathrm{Z}}\} = \overline{\vec{\tau}}\{\vec{\xi}/\vec{\mathrm{Z}}\}\{\vec{\gamma}/\vec{\mathrm{X}}\}.$$

and $\{\vec{\mathrm{Z}}\} \cap \mathsf{ftv}(\vec{\tau}\{\vec{\gamma}/\vec{\mathrm{X}}\}\{\vec{\xi}\{\vec{\gamma}/\vec{\mathrm{X}}\}/\vec{\mathrm{Z}}\}) = \emptyset$, using Lemma 26. Hence in fact:

$$\vdash_{\mathsf{0}} \overline{x}\langle\vec{y}\rangle \triangleright (x : \exists\vec{\mathrm{Z}}.(\vec{\tau})^{p_{\mathsf{0}}}, \vec{y} : \overline{\vec{\tau}\{\vec{\xi}/\vec{\mathrm{Z}}\}})\{\vec{\gamma}/\vec{\mathrm{X}}\}.$$

(PAR): Assuming $\vdash_{\phi_i} P_i \triangleright A_i$, $\phi_1 \asymp \phi_2$ and $A_1 \asymp A_2$, we get $A_1\{\vec{\gamma}/\vec{\mathrm{X}}\} \asymp A_2\{\vec{\gamma}/\vec{\mathrm{X}}\}$ and $A_1\{\vec{\gamma}/\vec{\mathrm{X}}\} \odot A_2\{\vec{\gamma}/\vec{\mathrm{X}}\}$ defined and equal to $(A_1 \odot A_2)\{\vec{\gamma}/\vec{\mathrm{X}}\}$ by Lemma 26. Hence $\vdash_{\phi_1 \odot \phi_2} P_1|P_2 \triangleright (A_1 \odot A_2)\{\vec{\gamma}/\vec{\mathrm{X}}\}$.

The remaining cases are simpler and mostly follow directly by the (IH). □

Next we formally define name substitution in action types: assuming that $\{\vec{v}\} \cap \{\vec{x}\} = \emptyset$, then $(\cdot)\{\vec{x}/\vec{v}\}$ is the least partial operation such that

- $A\{\epsilon/\epsilon\} = A$,
- $A\{x/v\} = A$, if $v \notin \mathsf{fn}(A)$,
- $(A, v : \tau)\{x/v\} = A \odot x : \tau$,
- $(A, v : \tau \to w : \sigma)\{x/v\} = A \odot x : \tau \to w : \sigma$, if $A \asymp x : \tau \to w : \sigma$ and $w \neq x$;
- $(A, w : \tau \to v : \sigma)\{x/v\} = A \odot w : \tau \to x : \sigma$, if $A \asymp w : \tau \to x : \sigma$ and $w \neq x$;

We extend the above definition to $A\{\vec{x}/\vec{v}\} = (..(A\{x_1/v_1\})..)\{x_n/v_n\}$.

Lemma 28. *1. If* $(A \odot B)\{x/v\}$ *is defined, then* $A\{x/v\}$ *and* $B\{x/v\}$ *are defined and* $A\{x/v\} \asymp B\{x/v\}$*, as well as* $A\{x/v\} \odot B\{x/v\} = (A \odot B)\{x/v\}$ *hold.*
2. If $A\{x/v\}$ *is defined and* a *is fresh, then* $(A, a : \tau)\{x/v\}$ *is also defined and equal to* $A\{x/v\}, a : \tau$.

Proof. By straightforward induction on the derivation of $(A \odot B)\{x/v\}$ we establish (1). (2) is straightforward. □

Lemma 29. *Let* $\vdash_\phi P \triangleright A$ *and assume* $A\{\vec{x}/\vec{v}\}$ *is defined. Then* $\vdash_\phi P\{\vec{x}/\vec{v}\} \triangleright A\{\vec{x}/\vec{v}\}$.

Proof. We shall establish if $A\{x/v\}$ is defined, then $\vdash_\phi P\{x/v\} \triangleright A\{x/v\}$ by induction on the derivation of $\vdash_\phi P \triangleright A$.

- (PAR): Let $\vdash_{\phi_i} P_i \triangleright A_i$, $\phi_1 \asymp \phi_2$ and $A_1 \asymp A_2$. As by assumption $(A_1 \odot A_2)\{x/v\}$ is defined, we can use Lemma 28 (1) and the (IH) to get to $\vdash_{\phi_1 \odot \phi_2} (P_1|P_2)\{x/v\} \triangleright A_1 \odot A_2\{x/v\}$.
- (RES): Immediate from Lemma 28 (2) and the (IH).
- (WEAK): Assume $\vdash_\phi P \triangleright A, y : \tau$ follows from $\vdash_\phi P \triangleright A^{-y}$ and $\mathsf{md}(\tau) \in \{\updownarrow, \mathbf{?}\}$. Let $(A, y : \tau)\{x/v\} = A \odot x : \tau$ be defined. If $y = v$ we have two subcases: (1) $x \in \mathsf{fn}(A)$, then, by modes, $A \odot x : \tau = A$ so we just use the (IH); otherwise (2) $A \odot x : \tau = A, x : \tau$ and we use the (IH) with a subsequent application of (WEAK). If $x = y \neq v$, we proceed as in (1). The remaining case, $y \neq v, x \neq y$ is like (2).
- (IN$^\downarrow$): Assume we derive $\vdash_{\mathtt{I}} x(\vec{v}).P \triangleright x : \forall\vec{\mathrm{x}}.(\vec{\tau})^\downarrow \to A, B$ from $\vdash_{\mathtt{O}} P \triangleright \vec{v} : \vec{\tau}, A, B$, with the usual restrictions on A and B. From the (IH) and the (VC) we get $\vdash_{\mathtt{O}} P\{y/w\} \triangleright \vec{v} : \vec{\tau}, (A, B)\{y/w\}$. The interesting case is where w does occur in A, B. If w is a name in A, it cannot be x, but also $y \neq x$ for otherwise the substitution would not be defined. So we apply (IN$^\downarrow$). Similarly, if w is a name in B.

The remaining rules are similar to the previous cases. □

C Proof of Proposition 5 (2)

We use the acyclicity of name, Proposition 3. Base cases $\overline{b}\langle\vec{y}\rangle$ is obvious. $b(\vec{y}).P$, $!b(\vec{y}).P$ and $(\boldsymbol{\nu}\, x)P$, are straightforward by induction on P (note that inputs and outputs always have connected types). We write $P_{\langle x:\tau\rangle}$ for a connected processes with type τ. Assume by induction typable term P' is translated into the bigger size of connected term, $P_{\langle x:\tau\rangle}$. We show a parallel composition $Q \mid P_{\langle x:\tau\rangle}$ with non-connected type is translated into the bigger size of connected terms.

Case (1) $Q \equiv \overline{z}\langle\vec{y}\rangle$ and $\mathsf{md}(\tau) = \mathbf{!}$. Then $(\boldsymbol{\nu}\, x)(Q \mid P)$ is connected.

Case (2) $Q \equiv \overline{z}\langle\vec{y}\rangle$ where z's mode is $\mathbf{?}$ and $\mathsf{md}(\tau) =\downarrow$ and $y_n = x$. Then $(\boldsymbol{\nu}\, x)(Q \mid P)$ is connected.

Case (3) $Q \equiv \overline{z}\langle\vec{y}\rangle$ where z's mode is $\mathbf{?}$ and $\mathsf{md}(\tau) =\downarrow$ and $y_n \neq x$. Assume P's linear output is c. Then $(\boldsymbol{\nu}\, y_n x)(Q \mid y_n(\vec{w}).\overline{x}\langle\vec{v}\rangle \mid P)$ is connected.

Case (4) $Q \equiv \overline{z}\langle\vec{y}\rangle$ where z's mode is $\uparrow$ and $\mathsf{md}(\tau) =\downarrow$ and $z = x$. Same as Case (2).

Case (5) $Q \equiv \overline{z}\langle\vec{y}\rangle$ where z's mode is $\uparrow$ and $\mathsf{md}(\tau) =\downarrow$ and $z \neq x$. Same as Case (3).

Case (6) $Q \equiv !a(\vec{y}).R$ and $\mathsf{md}(\tau) = \mathbf{!}$. Then by acyclicity of names, we know either (a) $a \in \mathsf{fn}(P)$ or (b) $x \in \mathsf{fn}(R)$ or (c)$a, x \notin \mathsf{fn}(P) \cup \mathsf{fn}(R)$. Assume (a). Then $(\boldsymbol{\nu}\, a)(Q \mid P)$ is connected. The case (b) is symmetric to (a). For the case (c), both $(\boldsymbol{\nu}\, a)(Q \mid P)$ and $(\boldsymbol{\nu}\, x)(Q \mid P)$ are connected.

Case (7) $Q \equiv a(\vec{y}).R$ and $\mathsf{md}(\tau) = \mathbf{!}$. Then $(\boldsymbol{\nu}\, x)(Q \mid P)$ is connected.

Case (8) $Q \equiv !a(\vec{y}).R$ and $\mathsf{md}(\tau) = \downarrow$. Then $(\boldsymbol{\nu}\, a)(Q \mid P)$ is connected.

Case (9) $Q \equiv a(\vec{y}).R$ and $\mathsf{md}(\tau) = \downarrow$. Then the case (a) $a \in \mathsf{fn}(P)$ is the same as (a) in Case (7), while the case (b) $x \in \mathsf{fn}(R)$ is as the same as (b) in Case (7). For the case (c) $a, x \notin \mathsf{fn}(P) \cup \mathsf{fn}(R)$, assume c' is a linear name in $\mathsf{fn}(R)$. Then $(\boldsymbol{\nu}\, c'x)(Q \mid c'(\vec{w}).\overline{x}\langle\vec{v}\rangle \mid P)$ is connected. $\square$

D Proofs in Section 5

D.1 Proof of Subject Reduction (Proposition 11)

We first prove the subject reduction theorem (Proposition 11) for the typing system in Figure 2.

Lemma 30. *1. If $A \asymp B$ then $A\{\tau/\mathrm{X}^{\exists}_{\tau}\} \asymp B\{\tau/\mathrm{X}^{\exists}_{\tau}\}$ and $A\{\tau/\mathrm{X}^{\exists}_{\tau}\} \odot B\{\tau/\mathrm{X}^{\exists}_{\tau}\} = (A \odot B)\{\tau/\mathrm{X}^{\exists}_{\tau}\}$. The corresponding result holds for universally annotated type variables.*

2. Assume $\vdash_{\phi} P \triangleright A$. Then $\vdash_{\phi} P \triangleright A\{\tau/\mathrm{X}^{\exists}_{\tau}\}$ for some τ such that $\mathsf{ftv}^{\exists}(\tau) = \emptyset$.

3. If $\vdash_{\phi} P \triangleright A$ and $\mathsf{ftv}(A) = \{\vec{\mathrm{X}}^{\forall}\}$, then $\vdash_{\phi} P \triangleright A\{\vec{\mathrm{X}}/\vec{\mathrm{X}}^{\forall}\}$ is derivable in the system of Figure 1.

4. If $\vdash_{\phi} P \triangleright A$ is derivable in the system of Figure 1, then $\vdash_{\phi} P \triangleright A\{\vec{\mathrm{X}}^{\forall}/\vec{\mathrm{X}}\}$

5. If $\vdash_{\phi} P \triangleright A$ and $P \equiv Q$, then also If $\vdash_{\phi} Q \triangleright A$.

6. If $\vdash_{\phi} P \triangleright A$ is derivable, then it is also derivable without using (∀-Var).

Proof. For (6) we also proceed by induction on the number of applications of (∀-Var). The key insight is that for (∀-Var) to have an effect, it needs universally annotated type variables. They can only have been introduced in (Weak) or (Out). But every time we use such a type variable in these rules, we use the appropriate concrete type τ instead. $\square$

By the above lemma, we can prove the same substitution lemma as in Lemma 3. Then we use Lemma 30 (1) to remove all existential variables and Lemma 30 (2) to convert universally annotated variables. Then we can type something like $\vdash_{\phi} P \triangleright A\{\vec{\tau}/\vec{\mathrm{X}}^{\exists}_{\tau}\}\{\vec{\mathrm{X}}/\vec{\mathrm{X}}^{\forall}\}$ in the system of Figure 1. Using the Subject Reduction Theorem 1, we have $\vdash_{\phi} Q \triangleright A\{\vec{\tau}/\vec{\mathrm{X}}^{\exists}_{\tau}\}\{\vec{\mathrm{X}}/\vec{\mathrm{X}}^{\forall}\}$ in the system of Figure 1. Then we apply (4) to obtain $\vdash_{\phi} Q \triangleright A\{\vec{\tau}/\vec{\mathrm{X}}^{\exists}_{\tau}\}$. Using (∃-Var) we finally get $\vdash_{\phi} Q \triangleright A$ as required. $\square$

D.2 Proofs for Subject Transition (Proposition 12)

In this subsection, we prove Proposition 12. We start from the small lemma.

Lemma 31. *Let* $\vdash_\phi P \triangleright A$. *Let* $A' = A$, *provided* $\mathsf{md}(A(x)) = \mathbf{?}$, *else* $A' = A \setminus x$,

1. *If* $\vdash_{\mathsf{0}} P \triangleright A \overset{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}{\rightsquigarrow} \vdash_{\mathtt{I}} Q \triangleright B$, *then* $B = \odot_{j\in J}(y_j : \overline{\sigma}_j \to a_j : \sigma_j) \odot_{k\in K} y_k : \overline{\sigma}_k$, *where* $\{\vec{a}\} \subseteq \mathsf{fn}(A)$, $\vec{y} = \langle y_0 ... y_{n-1}\rangle$, J *and* K *partition* $\{0, ..., n-1\}$ *and* $\sigma_i = \mathsf{con}(A(z_i)) = \mathsf{con}(A(y_i))$.
2. *If* $\vdash_{\mathtt{I}} P \triangleright A \overset{x\langle(\nu\vec{y})\vec{z}\rangle}{\rightsquigarrow} \vdash_{\mathsf{0}} Q \triangleright B$, *then* $B = A' \odot \vec{z} : \vec{\tau}$.

Now we prove Proposition 12. Clearly we only need to establish this for pretransitions. By induction on the derivation of the transition. We begin with (Com). Now we induce on the derivation of the two premises. Nested into these inner inductions, we do induction on the derivation of the typing judgements of the sources of the premise transitions.

Assume $\vec{\tau}$ is a vector of concrete types, i.e. no type variables. Set $\vec{\tau}' = \vec{\tau}\{\vec{\mathrm{X}}^{\forall}/\vec{\mathrm{X}}^{\exists}\}$. Let

$$\begin{aligned}
A_1 &= x : \exists\vec{\mathrm{X}}.(\vec{\tau}\vec{\mathrm{Y}}^{\forall})^{\uparrow}, \vec{y} : \overline{\vec{\sigma}}, \vec{z} : \overline{\vec{\mathrm{Y}}^{\forall}} \\
A_2 &= x : B, \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}'\overline{\vec{\mathrm{Y}}}^{\forall})^{\downarrow} \to A \\
A_2' &= \vec{v} : \overline{\vec{\tau}}', \vec{w} : \overline{\vec{\mathrm{Y}}}^{\forall}, \uparrow A^{-x}, \mathbf{?}B^{-x}
\end{aligned}$$

Assume that $A_1 \asymp A_2$, let $\vec{\rho}$ be another vector of concrete types and let $\vec{\mathrm{Y}}^{\forall}$ be a vector of of matching length. Then

$$\begin{aligned}
A_1\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} &= x : \exists\vec{\mathrm{X}}.(\vec{\tau}\vec{\rho})^{\uparrow}, \vec{y} : \overline{\vec{\sigma}}, \vec{z} : \overline{\vec{\rho}} \\
A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} &= x : B\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}, \forall\vec{\mathrm{X}}.(\overline{\vec{\tau}}'\overline{\vec{\rho}}^{\forall})^{\downarrow} \to A.
\end{aligned}$$

We don't need to apply $\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}$ to A because $\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}$ does not affect linear channels. Now assume that neither $A_1\{\vec{\rho}/\vec{\mathrm{Y}}\}$ nor $A_2\{\vec{\rho}/\vec{\mathrm{Y}}\}$ have any free type variables. This means we can derive

$$\dfrac{\dfrac{\sigma_i = \tau_i\{\vec{\gamma}/\vec{\mathrm{X}}^{\exists}\} \quad \mathsf{ftv}^{\exists}(\vec{\sigma}, \vec{\rho}) = \emptyset \quad \mathrm{X}_i \notin \mathsf{ftv}(\vec{\sigma})}{\vdash_{\mathsf{0}} \overline{x}\langle\vec{y}\vec{z}\rangle \triangleright A_1\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}} \qquad D = \vec{a} : \overline{\vec{\sigma}} \to \vec{y} : \vec{\sigma}, E = \vec{b} : \overline{\vec{\rho}} \to \vec{z} : \vec{\rho}}{\vdash_{\mathsf{0}} \overline{x}\langle\vec{y}\vec{z}\rangle \triangleright A_1\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \overset{\overline{x}\langle(\nu\vec{a}\vec{b})\vec{a}\vec{b}\rangle}{\rightsquigarrow} \vdash_{\mathtt{I}} \Pi[a_i \to y_i]^{\sigma_i} \mid \Pi[b_i \to z_i]^{\rho_i} \triangleright D, E}$$

Similarly, one derives

$$\dfrac{\dfrac{\vdash_{\mathsf{0}} P \triangleright A_2'\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \qquad \mathrm{X}_i^{\forall} \notin \mathsf{ftv}(A, B)}{\vdash_{\mathtt{I}} x(\vec{v}\vec{w}).P \triangleright A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}} \qquad A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \odot (\vec{a} : \vec{\tau}', \vec{b} : \vec{\rho}) \vdash x\langle(\nu\vec{a}\vec{b})\vec{a}\vec{b}\rangle}{\vdash_{\mathtt{I}} x(\vec{v}\vec{w}).P \triangleright A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \overset{x\langle(\nu\vec{a}\vec{b})\vec{a}\vec{b}\rangle}{\rightsquigarrow} \vdash_{\mathsf{0}} P\{\vec{a}\vec{b}/\vec{v}\vec{w}\} \triangleright A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}})}$$

Hence we can apply (Com) to obtain

$$
\frac{\vdash_{\mathtt{0}} \overline{x}\langle\vec{y}\vec{z}\rangle \triangleright A_1\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \overset{\overline{x}\langle(\nu\vec{a}\vec{b})\vec{a}\vec{b}\rangle}{\rightsquigarrow} \vdash_{\mathtt{I}} \Pi[a_i \to y_i]^{\sigma_i} \mid \Pi[b_i \to z_i]^{\rho_i} \triangleright D, E \qquad \vdash_{\mathtt{I}} x(\vec{v}\vec{w}).P \triangleright A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \overset{x\langle(\nu\vec{a}\vec{b})\vec{a}\vec{b}\rangle}{\rightsquigarrow} \vdash_{\mathtt{0}} P\{\vec{a}\vec{b}/\vec{v}\vec{w}\} \triangleright A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}})}{\vdash_{\mathtt{0}} \overline{x}\langle\vec{y}\vec{z}\rangle \mid x(\vec{v}\vec{w}).P \triangleright A_1 \odot A_2 \overset{\tau}{\rightsquigarrow} \vdash_{\mathtt{0}} (\nu\vec{a}\vec{b})(\Pi[a_i \to y_i]^{\sigma_i} \mid \Pi[b_i \to z_i]^{\rho_i} \mid P\{\vec{a}\vec{b}/\vec{v}\vec{w}\}) \triangleright A_1 \odot A_2}
$$

We have $B \asymp \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}}$ and $A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \asymp (\vec{a} : \vec{\tau}', \vec{b} : \vec{\rho})$ because all the names in $\vec{a}$ and $\vec{b}$ are fresh. Using the outermost (IH), we know that $\vdash_{\mathtt{I}} \Pi[a_i \to y_i]^{\sigma_i} \mid \Pi[b_i \to z_i]^{\rho_i} \triangleright D, E$ and $\vdash_{\mathtt{0}} P\{\vec{a}\vec{b}/\vec{v}\vec{w}\} \triangleright A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}})$. Now $A_1 \asymp A_2$ implies $A_1\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\} \asymp A_2\{\vec{\rho}/\vec{\mathrm{Y}}^{\forall}\}$ by Lemma 30 (1) . This in turn implies $D, E \asymp A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}})$. Let $Q = \Pi[a_i \to y_i]^{\sigma_i} \mid \Pi[b_i \to z_i]^{\rho_i}$. Then we can form

$$
\dfrac{\dfrac{\dfrac{\vdash_{\mathtt{I}} Q \triangleright D, E \qquad \vdash_{\mathtt{0}} P\{\vec{a}\vec{b}/\vec{v}\vec{w}\} \triangleright A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}})}{\vdash_{\mathtt{0}} Q \mid P\{\vec{a}\vec{b}/\vec{v}\vec{w}\} \triangleright (D, E) \odot (A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}}))}\ (\textsc{Par})}{\vdash_{\mathtt{0}} (\nu\vec{a}\vec{b})(Q \mid P\{\vec{a}\vec{b}/\vec{v}\vec{w}\}) \triangleright (\vec{y} : \vec{\sigma}, \vec{z} : \vec{\rho}) \odot (A, (B \odot \vec{a} : \overline{\vec{\tau}}', \vec{b} : \overline{\vec{\rho}}))}\ (\textsc{Res})}{\vdash_{\mathtt{0}} (\nu\vec{a}\vec{b})(Q \mid P\{\vec{a}\vec{b}/\vec{v}\vec{w}\}) \triangleright A_1 \odot A_2}\ (\textsc{Weak})
$$

as required. If (Res), (Weak) has been used to type the source of one of the premise transitions, we proceed straightforwardly by one of the (IHs). This leaves ($\exists$-Var). It impossible that we introduce an existential type variable that affects the types of any free name occurring in the transitions under induction. But then we can simply proceed by (IH).

The base case for replicated input is similar and the remaining cases of the innermost inductions (Res), (Par), (Weak) and ($\exists$-Var) are straightforward from the (IH). □

E An Alternative Generic Transition System

Figure 4 gives an alternative account of generic labelled transitions. It is easier to define than that of Section 5 as it doesn't require pretransitions or action predicates. But there's a price to be paid for simplicity: the inference system is not entirely compositional. Certain inputs can only be derived from non-trivial configurations.

Proposition 28. *Up to $\equiv$, transitions derivable using the system in Figure 3 coincide with those derivable using the system in Figure 4.*

Proof. By straightforward, yet tedious inductions on the derivations of transitions. □

$$
(\textsc{In}^{\downarrow}) \quad \frac{A(x) = \forall\vec{\mathrm{x}}.(\vec{\tau})^{\downarrow} \quad z_i \notin \{\vec{y}\} \text{ iff } A(z_i) = \Upsilon_i^{\exists} = \overline{\tau_i}}{\vdash_{\mathtt{I}} P \mid x(\vec{v}).Q \triangleright A \xrightarrow{x\langle(\nu\vec{y})\vec{z}\rangle} \vdash_{\mathtt{O}} P \mid Q\{\vec{z}/\vec{v}\} \triangleright A/x \odot \vec{z}:\vec{\tau}}
$$

$$
(\textsc{In}^{!}) \quad \frac{A(x) = \forall\vec{\mathrm{x}}.(\vec{\tau})^{!} \quad z_i \notin \{\vec{y}\} \text{ iff } A(z_i) = \Upsilon_i^{\exists} = \overline{\tau_i}}{\vdash_{\mathtt{I}} P \mid !x(\vec{v}).Q \triangleright A \xrightarrow{x\langle(\nu\vec{y})\vec{z}\rangle} \vdash_{\mathtt{O}} P \mid Q\{\vec{z}/\vec{v}\} \mid !x(\vec{v}).Q \triangleright A \odot \vec{z}:\vec{\tau}}
$$

$$
(\textsc{Out}) \quad \frac{A(x) = \exists\vec{\mathrm{x}}.(\vec{\tau})^{p\mathsf{O}} \quad w_i\{\vec{z}/\vec{y}\} \notin \{\vec{y}\} \text{ iff } A(w_i\{\vec{z}/\vec{y}\}) = \Upsilon_i^{\forall} = \overline{\tau_i} \quad \sigma_i = \mathsf{con}(\rho_i)}{\vdash_{\mathtt{O}} \overline{x}\langle\vec{w}\rangle \triangleright A \xrightarrow{\overline{x}\langle(\nu\vec{z})\vec{w}\{\vec{z}/\vec{y}\}\rangle} \vdash_{\mathtt{I}} \Pi_i[z_i \to y_i]^{\sigma_i} \triangleright \vec{z} : \overline{\vec{\sigma}} \to \vec{y} : \vec{\sigma}}
$$

$$
(\textsc{Com}) \quad \frac{}{\vdash_{\mathtt{O}} \overline{x}\langle\vec{y}\rangle \mid x(\vec{y}).P \triangleright x :\updownarrow, A \xrightarrow{\tau} \vdash_{\mathtt{O}} P\{\vec{y}/\vec{v}\} \triangleright x :\updownarrow, A}
$$

$$
(\textsc{Rep}) \quad \frac{}{\vdash_{\mathtt{O}} \overline{x}\langle\vec{y}\rangle \mid !x(\vec{y}).P \triangleright A \xrightarrow{\tau} \vdash_{\mathtt{O}} P\{\vec{y}/\vec{v}\} \mid !x(\vec{y}).P \triangleright A}
$$

$$
(\textsc{Par}) \quad \frac{\vdash_{\phi} P_1 \triangleright A_1 \xrightarrow{l} \vdash_{\psi} P_1' \triangleright A_1' \quad \vdash_{\mathtt{I}} P_2 \triangleright A_2 \quad A_1 \asymp A_2 \quad l \text{ allowed by } A_2}{\vdash_{\phi} P_1 | P_2 \triangleright A_1 \odot A_2 \xrightarrow{l} \vdash_{\psi} P_1' | P_2 \triangleright A_1' \odot A_2}
$$

$$
(\textsc{Res}) \quad \frac{\vdash_{\phi} P \triangleright A \xrightarrow{l} \vdash_{\psi} Q \triangleright B \quad x \notin \mathsf{fn}(l)}{\vdash_{\phi} (\boldsymbol{\nu}\, x)P \triangleright A/x \xrightarrow{l} \vdash_{\psi} (\boldsymbol{\nu}\, x)Q \triangleright B/x}
$$

$$
(\textsc{Cong}) \quad \frac{P \equiv P' \quad \vdash_{\phi} P' \triangleright A \xrightarrow{l} \vdash_{\phi} Q' \triangleright A \quad Q' \equiv Q}{\vdash_{\phi} P \triangleright A \xrightarrow{l} \vdash_{\phi} Q \triangleright A}
$$

Fig. 4. An alternative inference system for generic labelled transitions.