Sequentiality and the π-Calculus*

Martin Berger†, Kohei Honda†, and Nobuko Yoshida‡

Abstract.. We present a type discipline for the π-calculus which precisely captures the notion of sequential functional computation as a specific class of name passing interactive behaviour. The typed calculus allows direct interpretation of both call-by-name and call-by-value sequential functions. The precision of the representation is demonstrated by way of a fully abstract encoding of PCF. The result shows how a typed π-calculus can be used as a descriptive tool for a significant class of programming languages without losing the latter's semantic properties. Close correspondence with games semantics and process-theoretic reasoning techniques are together used to establish full abstraction.

1 Introduction

This paper studies a type discipline for the π-calculus which precisely captures the notion of sequential functional computation. The precision of the representation is demonstrated by way of a fully abstract encoding of PCF. Preceding studies have shown that while operational encodings of diverse programming language constructs into the π-calculus are possible, they are rarely fully abstract [34, 39]: we necessarily lose information by such a translation. This is because the translation of a source term M will generally result in a process containing superfluous behaviour. Type disciplines for the π-calculus with significant properties such as linearity and deadlock-freedom have been studied before [8, 17, 26, 27, 35, 37, 44]; however, to our knowledge, no previous typing system for the π-calculus has enabled a fully abstract translation of functional sequentiality. The present work shows that a relatively simple typing system suffices for this purpose. Despite its simplicity, the type discipline is general enough to give clean representations of both call-by-name and call-by-value sequentiality, offering a basic articulation of functional sequentiality without relying on particular evaluation strategies.

The core idea of the typing system is that *affineness* and *stateless replication* ensure deterministic computation. Sequentiality is guaranteed by controlling the number of threads through restricting the shape of processes. While the idea itself is simple, the result would offer a technical underpinning for the potential use of typed π-calculi as meta-languages for programming language study: having fully abstract descriptions in this setting means ensuring the results obtained in the

---

* Last modifiedon August 12, 2001. The extended abstract appeared in Proc. TLCA'01, LNCS, Springer, 2001. † Queen Mary, London, U.K. ‡ University of Leicester, Leicester, U.K.

meta-language to be transferable, in principle, to object languages. This opens the possibility to use the proposed typed syntax for the analysis of programming languages, especially when coupled with process-theoretic reasoning techniques.

From the viewpoint of the semantic study of sequentiality [5, 10, 33], our work positions sequentiality as a sub-class of the general universe of name passing interactive behaviour. This characterisation allows us to delineate sequentiality against the background of a broad computational universe which, among others, includes concurrency and non-determinism, offering a uniform basis on which various semantic findings can be integrated and extensions considered. Note sequential functional computation and its syntactic embodiment, the λ-calculus, have been the focus of diverse type structures and semantic structures which have been studied so far. By faithfully embedding a basic notion of sequentiality as a class of interacting processes preserving its type structure, the present work opens the possibility of using the π-calculus as a basic mathematical tool for the study of semantics and types of computation with the vast heritage from the studies on typed functional calculi and their semantics. This aspect is closely related with the lack of full abstraction in process encoding we already discussed: this lack indicates, at a deeper level, that the encoded types in the typed π-calculi guarantee only a weaker notion of behavioural properties than the original ones, so that the essential content of types is partially lost through the translation. The solution to this issue in the present work involves not only faithfulness in the embedding of the original type structure but also its generalisation, articulating a broader realm of typed functional behaviour. Not only, as mentioned above, can the new type structure conveniently encase both call-by-name and call-by-value sequentiality but also its simple extensions can capture other prominent notions of typed computation. As an example, [45] reports how a small change in the presented type structure allows the fully abstract embedding of the class of strongly normalising sequential computation. Other refinements, which can capture polymorphism, state and control, will be reported elsewhere (for preliminary accounts, see [19, 25]).

A significant point in this context is the close connection between the presented calculus and game semantics [3, 24, 28]: the structure of interaction of typed processes (with respect to typed environments) precisely conforms to the intensional structures of games introduced in [28] and studied in e.g. [2, 12, 24, 30, 32]. It is notable that the type discipline itself does not mention basic intensional notions in game semantics such as visibility, well-bracketing and innocence (although it does use a syntactic form of IO-alternation): yet they are derivable as operational properties of typed processes. We use this correspondence combined with process-theoretic reasoning techniques to establish full abstraction. Other forms of proofs would be possible: however the correspondence, in addition to facilitating the proof, offers deeper understanding of the present type discipline and game semantics.

We briefly give comparisons with related work. Hyland and Ong [29] presented a π-calculus encoding of innocent strategies of their games and show operational correspondence with a π-calculus encoding of PCF. Fiore and Honda

[12] propose another π-calculus encoding for call-by-value games [24]. Our work, while being built on these preceding studies, is novel in that it puts forward a general type discipline where typability ensures functional sequentiality. In comparison with game semantics, our approach differs as it is based on a syntactic calculus representing a general notion of computational behaviour, including concurrently interacting processes. As another difference in approach, game semantics starts from sequential behaviour, and relaxes its constraints to obtain other classes. Here the articulation goes in the other direction, starting from general, unconstrained behaviours to their specific subsets. All the more because of this difference, the coincidence of two characterisations would be a promising sign of their robustness. We also note that our results confirm some of the significant findings in game semantics, such as the equal status owned by call-by-name and call-by-value evaluation. From a different viewpoint, our work shows an effective way to apply game semantics to the study of basic typing systems for the π-calculus, in particular for the proof of full abstraction of encodings.

Concerning the use of the π-calculus as the target language for translations, [34] was the first to point out the difficulty of fully abstract embeddings of functional sequentiality in the π-calculus and [39] showed that the same problems arise even with the higher-order π-calculus. While some preceding work studies the significance of replication and linearity of channels [8, 17, 27, 35, 38, 41, 44], none offers a fully abstract interpretation of functional sequentiality. One of the novel features in this regard is the incorporation of duality into the type structures (cf. [14, 28]). This duality is closely related to the choice of syntax in our typed calculus, including the use of bound name passing.

There is a vast body of studies on the semantic characterisation of sequentiality. Apart from game semantics, there are a few recent work which treat a notion of sequentiality which incorporates control (which is related to Berry and Curien's sequential algorithms [5]). Cartwright, Curien, and Felleisen [9] present intensional characterisation of sequentiality with observable exceptions. More recently, Longley [31] studies classes of higher-order functions, first constructing them intensionally then studying them in terms of the internal structure of their extensional representation. Different representations would help the understanding of computational behaviour from different viewpoints. The approach based on π-calculus (or processes) would have a merit for uniformly capturing those classes of computation which may not be easily characterisable by functions, such as those with nondeterminism.

In the remainder, Section 2 and 3 introduce the typed calculus. Section 4 analyses operational structures of typed terms. Based on them Section 5 establishes full abstraction. The technical details, including proofs omitted from the main sections of the paper, are found in the appendices.

Acknowledgements. We thank Masato Hasegawa and Vasco Vasconcelos for their comments on the early version of the paper. Conversation with Masato Hasegawa, Luke Ong and Barry Jay deepened our understanding on the related topics. The first two authors are partially supported by EPSRC grant GR/N/37633. The last author is partially supported by EPSRC grant GR/R33465.

2 Processes

2.1 Syntax

We use a variant of the π-calculus as our base syntax. As in typed λ-calculi, we start from the leanest untyped syntax. The following gives the reduction rule of the asynchronous version of the π-calculus, introduced in [7, 21]:

$$x(\vec{y}).P \mid \overline{x}\langle\vec{v}\rangle \;\longrightarrow\; P\{\vec{v}/\vec{y}\} \tag{1}$$

Here $\vec{y}$ denotes a potentially empty vector $y_1...y_n$, $|$ denotes parallel composition, $x(\vec{y}).P$ is input, and $\overline{x}\langle\vec{v}\rangle$ is asynchronous output. Operationally, this reduction represents the consumption of an asynchronous message by a receptor. The idea extends to a replicated receptor $!\,x(\vec{y}).P$:

$$!\,x(\vec{y}).P \mid \overline{x}\langle\vec{v}\rangle \longrightarrow !\,x(\vec{y}).P \mid P\{\vec{v}/\vec{y}\}, \tag{2}$$

where the replicated process remains in the configuration after reduction.

Types for processes prescribe usage of names [35, 43]. To be able to do this with precision, it is important to control dynamic sharing of names. For this purpose it is essential to distinguish *free name passing* and *bound (private) name passing*: the latter allows tight control of sharing and can control name usage in more stringent ways. In the present study, using bound name passing alone is sufficient. Further, to have tractable inference rules, it is vital to specify bound names associated with the concerned output. Thus, instead of $(\boldsymbol{\nu}\,\vec{y})(\overline{x}\langle\vec{y}\rangle|P)$, we write $\overline{x}(\vec{y})\,P$, and replace (1) by the following reduction rule.

$$x(\vec{y}).P \,|\, \overline{x}(\vec{y})\,Q \;\longrightarrow\; (\boldsymbol{\nu}\,\vec{y})(P \,|\, Q) \tag{3}$$

Here "$\overline{x}(\vec{y})\,Q$" indicates that $\overline{x}(\vec{y})$ is an asynchronous output exporting $\vec{y}$ which are local to Q. The rule corresponding to (2) is given accordingly. To ensure asynchrony of outputs, we add the following rule to the standard closure rules for $|$ and $(\boldsymbol{\nu}\,x)$.

$$P \;\longrightarrow\; P' \quad\Rightarrow\quad \overline{x}(\vec{y})\,P \;\longrightarrow\; \overline{x}(\vec{y})\,P' \tag{4}$$

Further, the following structural rules are added to allow inference of interaction under an output prefix.

$$\overline{x}(\vec{z})\,(P|Q) \equiv (\overline{x}(\vec{z})\,P)|Q \qquad \text{if } \mathsf{fn}(Q) \cap \{\vec{z}\} = \emptyset, \tag{5}$$

$$\overline{x}(\vec{z})\,(\boldsymbol{\nu}\,y)P \equiv (\boldsymbol{\nu}\,y)\overline{x}(\vec{z})\,P \qquad \text{if } y \notin \{x, \vec{z}\}. \tag{6}$$

By these rules we maintain the dynamics based on the original asynchronous calculus (up to the equation $\overline{x}(\vec{z})\,P \equiv (\boldsymbol{\nu}\,\vec{z})(\overline{x}\langle\vec{z}\rangle|P)$), while enabling output actions to be typed with the same ease as input actions. Name-passing calculi using only bound name passing, called πI-calculi, have been studied in [6, 40].

Another useful construct for typing is *branching*. Branching is similar to the "case" construct in typed λ-calculi and can represent both base values such as booleans or integers and conditionals. While binary branching has some merit, we use indexed branching because it simplifies the description of base value passing. The branching variant of the reduction (3) becomes:

$$x[\&_{i\in I}(\vec{y_i}).P_i] \mid \overline{x}\mathtt{in}_j(\vec{y_j})\,Q \;\longrightarrow\; (\boldsymbol{\nu}\,\vec{y_j})(P_j \mid Q) \qquad (7)$$

where we assume $j \in I$, with $I\,(\neq \emptyset)$ denoting a finite or countably infinite indexing set. Accordingly we define the rule for replicated branching. Branching constructs of this kind have been studied in tyco [42] and other calculi [13, 16, 20] (the corresponding type structure already appeared in Linear Logic [1, 14]).

Augmenting the original asynchronous syntax with bound output and branching, we now arrive at the following grammar.

$$\begin{array}{lll|ll} P ::= & x(\vec{y}).P & \text{input} & P \mid Q & \text{parallel} \\ \mid & \overline{x}(\vec{y})\,P & \text{output} & (\boldsymbol{\nu}\,x)P & \text{hiding} \\ \mid & x[\&_{i\in I}(\vec{y_i}).P_i] & \text{branching input} & \mathbf{0} & \text{inaction} \\ \mid & \overline{x}\mathtt{in}_i(\vec{z})\,P & \text{selection} & !P & \text{replication} \end{array}$$

In $!P$ we require P to be either a unary or branching input. The bound/free names/variables are defined as usual and we assume the variable convention for bound names. The structural rules are standard except for the omission of $!P \equiv\, !P|P$ and the incorporation of (5) as well as (6) together with the corresponding rules for branching output. The reduction rules are as explained above, which also include variants of (3) and (7) for replicated branching inputs.

2.2 Examples

Henceforth we omit trailing zeros and null arguments and write $x[\&_i P_i]$ for $x[\&_i().P_i]$.

(i) $[\![\mathtt{n}]\!]_u \stackrel{\text{def}}{=}\, !\,u(a).\overline{a}\mathtt{in}_n$. Each time $[\![\mathtt{n}]\!]_u$ is invoked, it replies by telling its number, $\mathtt{n}$. Here a natural number becomes a *stateless server.*

(ii) $[\![\mathtt{succ}]\!]_u \stackrel{\text{def}}{=}\, !\,u(ya).\overline{y}(b)\,b[\&_n\,\overline{a}\mathtt{in}_{n+1}]$. $[\![\mathtt{succ}]\!]_x$ describes the behaviour of a successor function, which queries for its argument, a natural number as in (i) above, and returns its increment. This is another stateless server but this time it asks its client for an input.

(iii) $!u(xa).\overline{x}(zb)\,([\![\mathtt{1}]\!]_z \mid b[\&_i \overline{a}\mathtt{in}_i])$. This represents a type-2 functional $\lambda x.x1$: $(\mathsf{Nat}\Rightarrow\mathsf{Nat})\Rightarrow\mathsf{Nat}$. When the process is invoked, it queries for its argument (which is a function itself), that function then asks back for its own argument, to which $[\![\mathtt{1}]\!]_z$ replies. Finally the process receives, at b, an answer to its own question, based on which it answers to the initial question.

3 Typing

3.1 Action Modes

Functional computation is *deterministic.* There are two basic ways to realise this in interacting processes. One is to have (at most) one input and (at most) one output at a given channel (such a channel is called *affine*). Another is to have a unique stateless replicated input with zero or more dual outputs. These ideas have been studied in the past [14, 16, 17, 26, 27, 38, 41, 44]. To capture them in typing, we use the following *action modes*, denoted $p, q, \ldots$:

$!_1$	Affine input	$?_1$	Affine output
$!_\omega$	Replicated input	$?_\omega$	Output to replicated input

We also use $\perp$ to denote the presence of both input and output at an affine channel. In the table above, the mode on the left and that on the right in the same row are *dual* to each other, denoted $\overline{p}$ (for example, $\overline{!_1} = ?_1$).

3.2 Channel Types

Channel types indicate possible usage of channels. We use sorting [35] augmented with branching [1, 14, 16, 20, 42] and action modes. The grammar follows.

$$\begin{array}{ll} \alpha ::= \langle \tau,\ \overline{\tau} \rangle & \tau_{\mathtt{I}} ::= (\vec{\tau})^{!_1} \mid (\vec{\tau})^{!_\omega} \ \mid\ [\&_{i\in I}\, \vec{\tau_i}]^{!_1} \mid\ [\&_{i\in I}\, \vec{\tau_i}]^{!_\omega} \\ \tau ::= \tau_{\mathtt{I}} \ \mid\ \tau_{\mathtt{O}} & \tau_{\mathtt{O}} ::= (\vec{\tau})^{?_1} \mid (\vec{\tau})^{?_\omega} \mid\ [\oplus_{i\in I}\, \vec{\tau_i}]^{?_1} \mid\ [\oplus_{i\in I}\, \vec{\tau_i}]^{?_\omega} \end{array}$$

In the first line $\overline{\tau}$ denotes the *dual* of τ, which is the result of dualising all action modes and exchanging $\oplus$ and $\&$. A type of form $\langle \tau, \overline{\tau} \rangle$ is called *pair type*, which we regard as a set. $[\&_{i\in I} ...]$ corresponds to branching and $[\oplus_{i\in I} ...]$ corresponds to selection. As an example of types, let $\mathsf{Nat}^\bullet \stackrel{\text{def}}{=} [\oplus_{i\in\mathbb{N}}]^{?_1}$ and $\mathsf{Nat}^\circ \stackrel{\text{def}}{=} (\mathsf{Nat}^\bullet)^{!_\omega}$. Then in $!a(x).\overline{x}\mathtt{in}_n$, x is used as $\mathsf{Nat}^\bullet$ while a is used as Nat°.

A further idea in functional computation is asking a question and receiving a unique answer [3, 28]. A type is *sequential* when for each subexpression:[1]

(i) In $(\vec{\tau})^{!_\omega}$, there is at most one τ_i of mode $?_1$, with the rest being of mode $?_\omega$. Dually for $(\vec{\tau})^{?_\omega}$. The same applies to $[\&_{i\in I}\, \vec{\tau_i}]^{!_\omega}$ and $[\oplus_{i\in I}\, \vec{\tau_i}]^{?_\omega}$.

(ii) In $(\vec{\tau})^{!_1}$, each τ_i is of mode $?_\omega$, dually for $(\vec{\tau})^{?_1}$. The same applies to $[\&_{i\in I}\, \vec{\tau_i}]^{!_1}$ and $[\oplus_{i\in I}\, \vec{\tau_i}]^{?_1}$.

As an example, $(\overline{\mathsf{Nat}^\circ}\mathsf{Nat}^\bullet)^{!_\omega}$ is a sequential type for $[\![\mathsf{succ}]\!]_u$ in §2.2 (ii).

3.3 Action Types and IO-Modes

The sequents we use have the form $\Gamma \vdash_\phi P \triangleright A$. Γ is a *base*, i.e. a finite map from names to channel types, P is a process with type annotations on binding names, A is an *action type*, and ϕ is an *IO-mode*. Intuitively, an action type witnesses the real usage of channels in P with respect to their modes specified in Γ (thus controlling determinacy); an IO-mode ensures P contains at most one active thread (thus controlling sequentiality). Below in (i) we use a symmetric

[1] (i) below is a generalisation of what is given in [?]. This makes the type structure akin to those used in game semantics, thoughy no difference comes about as far as the main results in the paper go.

partial operator $\odot$ on action modes generated from $\mathbf{!}_1 \odot \mathbf{?}_1 = \bot$, $\mathbf{?}_\omega \odot \mathbf{?}_\omega = \mathbf{?}_\omega$ and $\mathbf{!}_\omega \odot \mathbf{?}_\omega = \mathbf{!}_\omega$. Thus, for example, $\mathbf{!}_\omega \odot \mathbf{!}_\omega$ is undefined. This partial algebra ensures that only one-one (resp. one-many) connection is possible at an affine (resp. replicated) channel.

(i) An *action type* assigns action modes to names. Each assignment is written px. $\mathsf{fn}(A)$ denotes the set of names in A. A partial operator $A \odot B$ is defined iff $p \odot q$ is defined whenever $px \in A$ and $qx \in B$; then we set $A \odot B = (A \backslash B) \cup (B \backslash A) \cup \{(p \odot q)x \mid px \in A, qx \in B\}$. We write $A \asymp B$ when $A \odot B$ is defined. The set of modes used in A is $\mathsf{md}(A)$.

(ii) An *IO-mode* is one of $\{\mathtt{I}, \mathtt{O}\}$. We set $\mathtt{I} \odot \mathtt{I} = \mathtt{I}$ and $\mathtt{I} \odot \mathtt{O} = \mathtt{O} \odot \mathtt{I} = \mathtt{O}$. Note $\mathtt{O} \odot \mathtt{O}$ is not defined. When $\phi_1 \odot \phi_2$ is defined we write $\phi_1 \asymp \phi_2$.

In IO-modes, $\mathtt{O}$ indicates a unique active output (consider it as a thread): thus $\mathtt{O} \not\asymp \mathtt{O}$ shows that we do not want more than one thread in a process.

3.4 Typing Rules

$$\text{(Zero)}\quad \frac{\Gamma\ \text{Sequential}}{\Gamma \vdash_{\mathtt{I}} \mathbf{0} \triangleright \emptyset}$$

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

$$\text{(Res)}\quad \frac{\Gamma \cdot x : \alpha \vdash_\phi P \triangleright A \otimes px \qquad p \in \{\bot, \mathbf{!}_\omega\}}{\Gamma \vdash_\phi (\boldsymbol{\nu}\, x : \alpha) P \triangleright A}$$

$$(\mathsf{In}^{\mathbf{!}_1}) \quad (C/\vec{y} = \mathbf{?}A) \quad \frac{\Gamma \vdash x : (\vec{\tau})^{\mathbf{!}_1} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{O}} P \triangleright C^{\text{-}x}}{\Gamma \vdash_{\mathtt{I}} x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \mathbf{!}_1 x}$$

$$(\mathsf{Out}^{\mathbf{?}_1}) \quad (C/\vec{y} = A \asymp \mathbf{?}_1 x) \quad \frac{\Gamma \vdash x : (\vec{\tau})^{\mathbf{?}_1} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P \triangleright C}{\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y} : \vec{\tau}) P \triangleright A \odot \mathbf{?}_1 x}$$

$$(\mathsf{Weak}\text{-}\bot) \quad \frac{\Gamma \vdash x : \mathbf{!}_1, \mathbf{?}_1 \qquad \Gamma \vdash_\phi P \triangleright A^{\text{-}x}}{\Gamma \vdash_\phi P \triangleright A \otimes \bot x}$$

$$(\mathsf{In}^{\mathbf{!}_\omega}) \quad (C/\vec{y} = \mathbf{?}_\omega A) \quad \frac{\Gamma \vdash x : (\vec{\tau})^{\mathbf{!}_\omega} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{O}} P \triangleright C^{\text{-}x}}{\Gamma \vdash_{\mathtt{I}} !\, x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \mathbf{!}_\omega x}$$

$$(\mathsf{Out}^{\mathbf{?}_\omega}) \quad (C/\vec{y} = A \asymp \mathbf{?}_\omega x) \quad \frac{\Gamma \vdash x : (\vec{\tau})^{\mathbf{?}_\omega} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P \triangleright C}{\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y} : \vec{\tau}) P \triangleright A \odot \mathbf{?}_\omega x}$$

$$(\mathsf{Weak}\text{-}\mathbf{?}_\omega) \quad \frac{\Gamma \vdash x : \mathbf{?}_\omega \qquad \Gamma \vdash_\phi P \triangleright A^{\text{-}x}}{\Gamma \vdash_\phi P \triangleright A \otimes \mathbf{?}_\omega x}$$

Fig. 1. Sequential Typing System

The typing rules are given in Figure 1. The rules for branching/selection are defined similarly and left to Appendix A. The following notation is used:

$\mathbf{?}_\omega A$	A s.t. $\mathsf{md}(A) = \{\mathbf{?}_\omega\}$	$A^{\text{-}x}$	A s.t. $x \notin \mathsf{fn}(A)$
$\mathbf{?}A$	A s.t. $\mathsf{md}(A) = \{\mathbf{?}_\omega, \mathbf{?}_1\}$	$A \otimes B$	$A \cup B$ s.t. $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \emptyset$
$A/\vec{x}$	$A \backslash \{p\vec{x}\}$ s.t. $\{\vec{x}\} \subseteq \mathsf{fn}(A)$	$\Gamma \cdot \Delta$	$\Gamma \cup \Delta$ s.t. $\mathsf{fn}(\Gamma) \cap \mathsf{fn}(\Delta) = \emptyset$

$\Gamma \vdash x\!:\!\tau$ denotes $x\!:\!\tau$ or $x\!:\!\langle\tau,\overline{\tau}\rangle$ in Γ, while $\Gamma \vdash x\!:\!p$ indicates $\Gamma \vdash x : \tau$ such that the mode of τ is p. Typed processes are often called *sequential processes.* The sequent $\Gamma \vdash_\phi P \triangleright A$ is often abbreviated to $\Gamma \vdash_\phi P$.

We briefly illustrate each typing rule. In (Zero), we start in $\mathtt{I}$-mode since there is no active output. In (Par), "$\asymp$" controls composability, ensuring that at most one thread is active in a given term. In (Res), we do not allow $?_1$, $?_\omega$ or $!_1$-channel to be restricted since these actions expect their dual actions exist in the environment (cf. [17, 22, 27]). ($\mathsf{In}^{!_1}$) ensures that x occurs precisely once (by $C^{\text{-}x}$) and no free input is suppressed under prefix (by $C/\vec{y} = ?A$). ($\mathsf{Out}^{?_1}$) also ensures an output at x occurs precisely once, but does not suppress the body by prefix since output is asynchronous (essentially the rule composes the output prefix and the body in parallel). (Weak-$\bot$) allows assigning the same type after a pair of dual affine channels disappears following an interaction. This is essential for subject reduction. ($\mathsf{In}^{!_\omega}$) is the same as ($\mathsf{In}^{!_1}$) except no free $?_1$-channels are suppressed (note that if a $?_1$-channel is under replication then it can be used more than once). ($\mathsf{Out}^{?_\omega}$) and (Weak-$?_\omega$) say $?_\omega$-channels occur zero or more times, and it does not suppress any actions. Finally, in ($\mathsf{Out}^{?_1}$) and ($\mathsf{Out}^{?_\omega}$), the premise must have $\mathtt{I}$-mode for otherwise we would end up with more than one thread. Note that, for input, we require the premise to be $\mathtt{O}$-mode. This together ensures single-threadedness to be invariant under reduction, as we discuss later.

3.5 Examples

The following examples indicate how the present type discipline imposes strong constraints on term structure.

(i) Given $\Gamma = a\!:\!()^{?_1}\!\cdot\! b\!:\!\langle()^{!_1},()^{?_1}\rangle \cdot c\!:\!()^{?_1}$, we build sequential processes one by one, starting from inaction. (1) $\Gamma \vdash_{\mathtt{I}} \mathbf{0} \triangleright \emptyset$, (2) $\Gamma \vdash_{\mathtt{O}} \overline{a} \triangleright ?_1 a$, and (3) $\Gamma \vdash_{\mathtt{I}} b.\overline{a} \triangleright ?_1 a \otimes !_1 b$. Then we have:

$$\Gamma \vdash_{\mathtt{O}} \overline{b} \mid b.\overline{a} \triangleright ?_1 a \otimes \bot b \quad \text{with} \quad ?_1 b \odot !_1 b = \bot b \quad \text{and} \quad \mathtt{O} \odot \mathtt{I} = \mathtt{O}$$

where "$\bot b$" means name b is no longer composable. Note for any ϕ, $\Gamma \not\vdash_\phi b.\overline{a} \mid b.\overline{c}$ since b is affine.

(ii) Given $\Gamma = a\!:\!()^{?_\omega}\!\cdot\! b\!:\!\langle()^{!_\omega},()^{?_\omega}\rangle$, we have:
- $\Gamma \vdash_{\mathtt{O}} \ \overline{a} \mid !b.\overline{a} \triangleright ?_\omega a \otimes !_\omega b$ with $?_\omega a \odot ?_\omega a = ?_\omega a$ and $\mathtt{O} \odot \mathtt{I} = \mathtt{O}$; and
- $\Gamma \vdash_{\mathtt{O}} !b.\overline{a} \mid \overline{b} \triangleright ?_\omega a \otimes !_\omega b$ with $?_\omega b \odot !_\omega b = !_\omega b$.

However, for any ϕ, $\Gamma \not\vdash_\phi \overline{a} \mid !b.\overline{a} \mid \overline{b}$ since $\mathtt{O} \odot \mathtt{O}$ is undefined. This example shows control by modes is essential even if $?_\omega$-mode channel does not appear in parallel; we can check after one step interaction between $!b.\overline{a}$ and $\overline{b}$, two messages to a will appear in parallel.

(iii) For $[\![\mathbf{n}]\!]_u$ in Example 2.2 (i), we have $u\!:\!\mathsf{Nat}^\circ \vdash_{\mathtt{I}} [\![\mathbf{n}]\!]_u$ (see § 3.2 for Nat°).

(iv) For $[\![\mathsf{succ}]\!]_u$ in Example 2.2 (ii), we can derive $u\!:\!(\overline{\mathsf{Nat}^\circ}\mathsf{Nat}^\bullet)^{!_\omega} \vdash_{\mathtt{I}} [\![\mathsf{succ}]\!]_u$.

(v) For the process in Example 2.2 (iii), let $\tau \stackrel{\text{def}}{=} ((\mathsf{Nat}^\circ\overline{\mathsf{Nat}^\bullet})^{?_\omega}\mathsf{Nat}^\bullet)^{!_\omega}$. Then we have $u : \tau \vdash_{\mathtt{I}} !u(xa).\overline{x}(zb)\,([\![\mathtt{1}]\!]_z \mid b[\&_{i\in\mathbb{N}}\overline{a}\mathbf{in}_i]) \triangleright !_\omega u$.

(vi) A *copy-cat* $[x \to y]^\tau \stackrel{\text{def}}{=} !x(a).\overline{y}(b)b.\overline{a}$ copies all behaviour starting at one channel to those starting at another. Let $\tau = (()^{?_1})^{!_\omega}$ and $\Gamma = x:\tau \cdot y:\overline{\tau}$. Then (1) $\Gamma \cdot a:()^{?_1} \cdot b:()^{!_1} \vdash_{\mathtt{I}} b.\overline{a} \triangleright ?_1 a \otimes !_1 b$, (2) $\Gamma \cdot a:()^{?_1} \vdash_{\mathtt{O}} \overline{y}(b)b.\overline{a} \triangleright ?_1 a \otimes ?_\omega y$, with $(?_1 a \otimes !_1 b)/b = ?_1 a$, and (3) $\Gamma \vdash_{\mathtt{I}} [x \to y]^\tau \triangleright !_\omega x \otimes ?_\omega y$.

Taking for example $(\boldsymbol{\nu}\, x)(P|[x \to y]^\tau)$ with $P \stackrel{\text{def}}{=} \overline{x}(a)a.\overline{c}$, we can check that all actions of P are copied from x to y (this does not include c which is emitted by P).

(vii) Let $\Delta = x:\langle\tau, \overline{\tau}\rangle \cdot y:\langle\tau, \overline{\tau}\rangle \cdot z:\langle\tau, \overline{\tau}\rangle$ and $\tau = (()^{?_1})^{!_\omega}$. Then we have:

- connection of two links: $\Gamma \vdash_{\mathtt{I}} [x \to y]^\tau \mid [y \to z]^\tau \triangleright !_\omega x \otimes !_\omega y \otimes ?_\omega z$ with $!_\omega y \otimes ?_\omega y = !_\omega y$.
- links to a shared resource at z: $\Gamma \vdash_{\mathtt{I}} [x \to z]^\tau \,|\, [y \to z]^\tau \triangleright !_\omega x \otimes !_\omega y \otimes ?_\omega z$ with $?_\omega z \otimes ?_\omega z = ?_\omega z$.

However, for any ϕ and environent, $[x \to z]^\tau \mid [x \to y]^\tau$ which represents non-deterministic forwarding is untypable since $!_\omega x \odot !_\omega x$ is undefined.

(viii) Let $\rho \stackrel{\text{def}}{=} ([\oplus_{i \in \mathbb{N}}]^{?_1})^{!_\omega}$ and $\Omega^\rho_z \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, xy)([x \to y]^\rho | [y \to x]^\rho | \overline{x}(a)\, a[\&_{i \in \mathbb{N}} \overline{z}\mathtt{in}_i])$. Then $u:\rho \vdash_{\mathtt{I}} !u(z).\Omega^\rho_z \triangleright !_\omega u$. Unlike $[\![\mathtt{n}]\!]_u$, it returns nothing when asked, representing the undefined.

3.6 Basic Syntactic Properties

The type discipline satisfies the following standard properties. In (i) and (ii) below, the partial order $\leq$ on bases is generated from set inclusion and the rule $\Gamma \leq \Delta \;\Rightarrow\; \Gamma \cdot x:\tau \leq \Delta \cdot x:\langle\tau, \overline{\tau}\rangle$. The order on action types is simply set inclusion. In (iii) we let $\twoheadrightarrow \stackrel{\text{def}}{=} \equiv \cup (\longrightarrow)^*$.

Proposition 1. (i) (weakening) *If $\Delta \leq \Gamma$ and $\Delta \vdash_\phi P$ then $\Gamma \vdash_\phi P$.*

(ii) (minimal type) *A typable process has a minimum base and action type. Further, if $\Gamma \vdash_\phi P$ and $\Delta \vdash_\psi P$ then $\phi = \psi$.*

(iii) (subject reduction) *If $\Gamma \vdash_\phi P$ and $P \twoheadrightarrow Q$ then $\Gamma \vdash_\phi Q$.*

We say an occurrence (subterm) in a process is an *active input* (resp. *active output*) if it is an input-prefixed (resp. output-prefixed) term which neither occurs under an input prefix nor has its subject bound by an output prefix.

Proposition 2. (i) *Let $\Gamma \vdash_\phi P \triangleright A \otimes px$ such that $p \in \{!_\omega, !_1\}$. Then there is an active input with free subject x in P.*

(ii) *Let $\Gamma \vdash_\phi P$. (1) If $\phi = \mathtt{I}$ there is no active output in P; (2) If $\phi = \mathtt{O}$ there is a unique active output in P; and (3) In both cases, two input processes never share the same name for their subjects, either bound or free.*

Corollary 1. (determinacy) *If $\Gamma \vdash_\phi P$ and $P \longrightarrow Q_i$ $(i = 1, 2)$ then $Q_1 \equiv Q_2$ and $\phi = \mathtt{O}$.*

3.7 Contextual Equality

Corollary 1 suggests non-deterministic state change (which plays a basic role in e.g. bisimilarity and testing/failure equivalence) may safely be ignored in typed equality, so that a Morris-like contextual equivalence suffices as a basic equality over processes. Let us say x is *active* when it is the free subject of an active input/output, e.g. x in $(\boldsymbol{\nu}\,\vec{w})(\overline{x}(\vec{y})P \mid R)$ assuming $x \notin \vec{w}$. We first define:

$$\Gamma \vdash_{\phi} P \Downarrow_x \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \Gamma \vdash_{\phi} P \twoheadrightarrow P' \text{ with } x \text{ active in } P' \;\text{ and }\; \Gamma \vdash_{\phi} P \triangleright A \otimes \mathbf{?}_1 x.$$

Choosing only affine output as observables induces a strictly coarser (pre-)congruence than if we had also included non-affine output ($\mathbf{?}_\omega$-actions are not considered since, intuitively, they do not affect the environment). We can now define a typed equality. Below, a relation over sequential processes is *typed* if it relates only processes with identical base, action type and IO-mode. A relation $\cong \supseteq \equiv$ is a *typed congruence* when it is a typed equivalence closed under typed contexts and, moreover, it satisfies: if $\Gamma \geq \Delta$ and $\Delta \vdash_{\phi} P \cong Q$ then $\Gamma \vdash_{\phi} P \cong Q$.

Definition 1. $\cong_{\mathsf{seq}}$ is the maximum typed congruence on sequential processes such that: if $\Gamma \vdash_{\phi} P \cong_{\mathsf{seq}} Q$ and $\Gamma \vdash_{\phi} P \Downarrow_x$ then $\Gamma \vdash_{\phi} Q \Downarrow_x$.

While the use of $\Downarrow_x$ in Definition 1 may look ad hoc, this is not so. The status of the theory is clarified by the following characterisation. Let us say a typed term $\Gamma \vdash P$ is *insensitive* when $P \longrightarrow^* P'$ implies P' does not own neither free active input nor free active output. We then say a typed congruence $\cong'$ is *sound* (cf. [23]) iff: (1) it is a consistent equality (i.e. not universal) which includes $\equiv$; (2) it is reduction closed (i.e. $P \cong' Q$ and $P \longrightarrow P'$ implies for some Q' we have $Q \longrightarrow^* Q'$ and $P' \cong' Q'$) and, moreover, (3) it always equates insensitive terms with the same type. Then we can show, using the same method employed in [23], that the maximum sound congruence (exists and) coincides with $\cong_{\mathsf{seq}}$ (see Appendix D.3 for the outline of the proof). Thus $\cong_{\mathsf{seq}}$ (and the corresponding observable $\Downarrow_x$) arises canonically, just as the standard observational congruence on PCF terms arises as a maximal consistent equality of some kind.

4 Analysis of Sequential Interactive Behaviour

4.1 Preamble

The purpose of the rest of the paper is to demonstrate that our typed processes precisely characterise the notion of functional sequentiality. By functional sequentiality we mean the class of computational dynamics that is exhibited by, for example, call-by-name and call-by-value PCF. Concretely we show, via an interpretation $u : \alpha^\circ \vdash_{\mathtt{I}} [\![M_i : \alpha]\!]_u$ that, for a PCF term $\vdash M_i : \alpha$ $(i = 1, 2)$, we have $M_1 \cong M_2$ iff $u : \alpha^\circ \vdash_{\mathtt{I}} [\![M_1 : \alpha]\!]_u \cong_{\mathsf{seq}} [\![M_2 : \alpha]\!]_u$. Here $\cong$ is the standard contextual equality on PCF-terms [15]. To this end we first introduce typed transitions to give a tractable account of processes interacting in typed contexts (the latter, like the former, must be input-output alternating). We then show that these transitions satisfy central properties of the intensional structures of games introduced in [28], namely visibility, bracketing and innocence. In particular, by innocence, any sequential process is representable by the corresponding innocent function up to redundant τ-actions. Further, the typed behaviour of a composite process $P|Q$ is completely determined by that of P and Q. Finally we show, *à la* game semantics, that any difference between typed processes in $\cong_{\text{seq}}$ can be detected by sequential "tester" processes whose graphs as innocent functions are finite. But finite processes in (the interpretation of) PCF types are in turn representable by PCF-terms up to $\cong$, leading to the completeness of the interpretation. Since soundness is easy by operational correspondence, this establishes full abstraction. In the following we illustrate key steps of reasoning to reach finite definability.

Note on terminology. In this section, correspondence with typed transition and intensional structures of games is a central topic. Since there is some difference in terminology between process calculi and game semantics, we list the correspondence for reference.

O's Question (OQ)	[	$!_\omega$	P's Answer (PA)	]	$?_1$
P's Question (PQ)	(	$?_\omega$	O's Answer (OA)	)	$!_1$

Note that "O" is usually used to indicate "Opponent" in game semantics, which corresponds to *input* in our (process-algebraic) terminology. To avoid confusion, we shall consistently use "input" and "output" rather than "Opponent" and "Player".

4.2 Typed Transitions

Let $P \stackrel{\text{def}}{=} !x(yz).\overline{y}(c)c.\overline{z}$ and $Q \stackrel{\text{def}}{=} \overline{x}(yz)(!y(c).\overline{c}|z.\overline{w})$. Then $P|Q$ is well-typed, and we have:

$$\begin{aligned} P \mid Q &\longrightarrow (\boldsymbol{\nu}\, yz)((P|\overline{y}(c)c.\overline{z}) \mid (!y(c).\overline{c}|z.\overline{w})) \\ &\longrightarrow (\boldsymbol{\nu}\, yzc)((P|c.\overline{z}) \mid (\overline{c}|z.\overline{w}|!y(c).\overline{c})) \\ &\longrightarrow (\boldsymbol{\nu}\, yzc)((P|\overline{z}) \mid (z.\overline{w}|!y(c).\overline{c})) \\ &\longrightarrow (\boldsymbol{\nu}\, yzc)(P \mid (\overline{w}|!y(c).\overline{c})). \end{aligned}$$

This example suggests that input and output alternate in typed interaction. Indeed this is the only way sequential processes interact: if P does an output and Q does an input, then the derivatives of P and Q should now be in I-mode and O-mode, respectively. If they interact again, input and output are reversed. Typed transitions are built on this idea.

First we generate *untyped transitions* $P \stackrel{l}{\longrightarrow} Q$, with labels τ, $x(\vec{y})$, $\overline{x}(\vec{y})$, $x\mathtt{in}_i(\vec{y})$ and $\overline{x}\mathtt{in}_i(\vec{y})$ by the following rules.

(In)	$x(\vec{y}).P \xrightarrow{x(\vec{y})} P$	(Out)	$\overline{x}(\vec{z})P \xrightarrow{\overline{x}(\vec{z})} P$
(Bra)	$x[\&_{i\in I}(\vec{y_i}).P_i] \xrightarrow{x\mathtt{in}_i(\vec{y_i})} P_i$	(Sel)	$\overline{x}\,\mathtt{in}_i(\vec{z})P \xrightarrow{\overline{x}\mathtt{in}_i(\vec{z})} P$

The rules for replicated input are defined similarly. The contextual rules are standard except for closure under asynchronous output (we omit the corresponding rule for branching).

$$(\text{OUT-}\xi)\quad P \xrightarrow{l} P' \text{ with } \mathsf{fn}(l) \cap \{\vec{y}\} = \emptyset \quad\Rightarrow\quad \overline{x}(\vec{y})\,P \xrightarrow{l} \overline{x}(\vec{y})\,P'$$

To turn this into typed transitions, we first restrict the transitions of a process of mode $\mathtt{O}$ to only τ-actions and outputs since (as discussed at the outset) the interacting party should always be in $\mathtt{I}$-mode. Secondly, if a process has $\bot x$ (resp. $!_\omega x$) in its action type, then both input and output at x (resp. output at x) are excluded since, again, such actions can never be observed in a typed context. It is easy to check that sequential processes are closed under the restricted transition relation. The resulting typed transitions are written:

$$\Gamma \vdash_\phi P \xrightarrow{l} \Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_\psi Q$$

where $\vec{y}\!:\!\vec{\tau}$ assigns names introduced in l as prescribed by Γ. Typed τ-transitions coincide with untyped τ-transitions, hence typing of transitions restricts only observability of actions, not computation. Basic properties of transitions follow.

Proposition 3. (i) (IO-alternation) *Let* $\Gamma \vdash_\phi P \stackrel{l_1 l_2}{\Longrightarrow} \Delta \vdash_\psi Q$. *Then* (1) $\phi = \psi$, *and* (2) l_1 *is input iff* l_2 *is output and vice versa.*

(ii) (determinacy) *If* $\Gamma \vdash_\phi P \xrightarrow{l} \Delta \vdash_\psi Q_i$ $(i = 1,2)$ *then* $Q_1 \equiv_\alpha Q_2$.

(iii) (unique output) *If* $\Gamma \vdash_{\mathtt{O}} P \xrightarrow{l_i} P_i$ $(i = 1,2)$ *then* $l_1 \equiv_\alpha l_2$.

As an example of typed transitions, let $\tau \stackrel{\text{def}}{=} (\overline{\mathsf{Nat}^\circ \mathsf{Nat}^\bullet})^{!_\omega}$. Then, using the notation in Examples 2.2 (ii), we have:

$$\begin{array}{ll} x\!:\!\tau \vdash_{\mathtt{I}} [\![\mathtt{succ}]\!]_u \stackrel{u(ya)}{\longrightarrow} & u\!:\!\tau, y\!:\!\overline{\mathsf{Nat}^\circ}, a\!:\!\mathsf{Nat}^\bullet \vdash_{\mathtt{O}} \overline{y}(b)\, b[\&_{i\in\mathbb{N}}\, \overline{a}\mathtt{in}_{i+1}] \mid [\![\mathtt{succ}]\!]_u \\ \stackrel{\overline{y}(b)}{\longrightarrow} & u\!:\!\tau, y\!:\!\overline{\mathsf{Nat}^\circ}, a\!:\!\mathsf{Nat}^\bullet, b\!:\!\overline{\mathsf{Nat}^\bullet} \vdash_{\mathtt{I}} b[\&_{i\in\mathbb{N}}\, \overline{a}\mathtt{in}_{i+1}] \mid [\![\mathtt{succ}]\!]_u \\ \stackrel{b\mathtt{in}_j}{\longrightarrow} & u\!:\!\tau, y\!:\!\overline{\mathsf{Nat}^\circ}, a\!:\!\mathsf{Nat}^\bullet, b\!:\!\overline{\mathsf{Nat}^\bullet} \vdash_{\mathtt{O}} \overline{a}\mathtt{in}_{j+1} \mid [\![\mathtt{succ}]\!]_u \\ \stackrel{\overline{a}\mathtt{in}_{j+1}}{\longrightarrow} & u\!:\!\tau, y\!:\!\overline{\mathsf{Nat}^\circ}, a\!:\!\mathsf{Nat}^\bullet, b\!:\!\overline{\mathsf{Nat}^\bullet} \vdash_{\mathtt{I}} \mathbf{0} \mid [\![\mathtt{succ}]\!]_u \end{array}$$

4.3 Visibility and Well-Bracketing

Let us write $\Gamma \vdash_\phi P \stackrel{l_1..l_n}{\Longrightarrow} \Delta \vdash_\psi Q$ if $\Gamma \vdash_\phi P \xrightarrow{\tau^*}\xrightarrow{l_1}\xrightarrow{\tau^*} \ldots \xrightarrow{\tau^*}\xrightarrow{l_n}\xrightarrow{\tau^*} \Delta \vdash_\psi Q$ with $l_i \neq \tau$ $(0 \leq i \leq n)$. For $i \lneq j$, we write $l_i \curvearrowright_{\mathsf{b}} l_j$ (read: l_i binds l_j) when the subject of l_j is bound by l_i (e.g. $x(y) \curvearrowright_{\mathsf{b}} \overline{y}\mathtt{in}_n$). Clearly, in typable processes, input only binds output and vice versa. $\curvearrowright_{\mathsf{b}}$ corresponds to justification of moves in games. Now we define the notion of *views* as follows. $\ulcorner l_1 ... l_n \urcorner^{\mathtt{O}}$ is defined first, with $s, t, \ldots$ ranging over sequences of labels.

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

Input view, denoted $\ulcorner s \urcorner^{\mathtt{I}}$, is defined dually by exchanging $\mathtt{O}$s and $\mathtt{I}$s as well as input and output. We often confuse $\ulcorner s \urcorner^{\mathtt{O}}$ and $\ulcorner s \urcorner^{\mathtt{I}}$ with the corresponding sequences. We now define:

Definition 2. (visibility) Let $\Gamma \vdash_\phi P \stackrel{s}{\Longrightarrow} \Delta \vdash_\psi Q$. Then $s = l_1 \cdots l_n$ is *input-visible* if whenever l_{i+1} is input such that $l_j \curvearrowright_{\mathtt{b}} l_i$, we have $j \in \ulcorner l_1 \cdots l_i \urcorner^{\mathtt{I}}$. Dually we define output-visibility. We say $\Gamma \vdash_\phi P$ is *visible* if whenever $\Gamma \vdash_\phi P \stackrel{s}{\Longrightarrow}$ and s is input-visible then it is output-visible.

The first key result follows.

Proposition 4. *$\Gamma \vdash_\phi P$ is visible.*

The proof proceeds by first establishing that it suffices to consider only *well-knit* traces where the only free input (if any) is an initial one. We then use induction on the typing rules to show that well-knit traces are visible. The only non-trivial cases are input prefixes and parallel composition. For input prefixes we use Proposition 3.2 (i). For parallel composition, we use *composite transitions* of $\Gamma \vdash_\phi P|Q$ which record the transitions of P and Q contributing to the those of $P|Q$ as a whole. Such transitions can be written in a matrix with four rows. For example, a composite transition of a sequential process (omitting types) $!x(c).\overline{y}(e).e[\&_{i\in\mathbb{N}}\overline{c}\mathtt{in}_{i+1}] \mid !y(e).\overline{z}(e').e'[\&_{i\in\mathbb{N}}\overline{e}\mathtt{in}_i]$ is given as follows, writing P and Q for the first and second components of parallel composition:

$$
\begin{array}{lllllll}
P\text{-visible}: & x(c) & & & & & \overline{c}\mathtt{in}_3 \\
P\text{-}\tau: & & \overline{y}(e) & & & e\mathtt{in}_2 & \\
Q\text{-}\tau: & & y(e) & & & \overline{e}\mathtt{in}_2 & \\
Q\text{-visible}: & & & \overline{z}(e') & e'\mathtt{in}_2 & &
\end{array}
$$

If such a sequence is well-knit and input-visible in its observable part (i.e. the first and fourth rows), then it satisfies the *switching condition* [3, 28], i.e. the action of P (resp. Q) moving from one row to another is always an output. To establish this we use IO-modes of derivatives and input-visibility. Then output visibility is immediate using standard game semantics technique [24, 28, 32].

Next, *well-bracketing* [3, 28] says that later questions are always answered first, i.e. nesting of bracketing is always properly matched. Below, following the table in §4.1, we call actions of mode $\mathbf{!}_\omega$ and $\mathbf{?}_\omega$ *questions* while actions of mode $\mathbf{!}_1$ and $\mathbf{?}_1$ are *answers*.

Definition 3. Let $\Gamma \vdash_\phi P \stackrel{s}{\Longrightarrow} \Delta \vdash_\psi Q$ be input-visible. Then s is *well-bracketing* if, whenever $s' = s_0 \cdot l_i \cdot s_1 \cdot l_j$ for a prefix s' of s is such that (1) l_i is a question and (2) l_j is an answer free in $s_1 \cdot l_j$, we have $l_i \curvearrowright_{\mathtt{b}} l_j$.

Now we say $\Gamma \vdash_\phi P$ is *well-bracketing* if whenever $\Gamma \vdash_\phi P \stackrel{sl}{\Longrightarrow}$, s is well-bracketing and l is output, then sl is well-bracketing. Then we have:

Proposition 5. *$\Gamma \vdash_\phi P$ is well-bracketing.*

The proof uses induction on typing rules, noting that it suffices to consider well-knit sequences. The non-trivial cases are input by $!_\omega$ and parallel composition. The former holds because a $!_\omega$-prefix does not suppress a free output with action mode $?_1$, while the latter follows from the switching condition [24, 28, 32].

Definition 4. (legal trace) Let $\Gamma \vdash_\phi P \stackrel{s}{\Longrightarrow}$. Then s is *legal* if it is both input-visible and well-bracketing.

4.4 Innocence

Innocence [28] says that a process does the same action whenever it is in the same "context", i.e. in the same output-view. To establish innocence of traces of typed processes we begin with the following lemma, proved by analysis of possible redexes relying on the shape of the syntax imposed by the type discipline.

Lemma 1. (permutation) *Let* $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_1 l_2 l_3 l_4}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ *such that* $l_1 \not\curvearrowright_{\mathsf{b}} l_4$ *and* $l_2 \not\curvearrowright_{\mathsf{b}} l_3$. *Then* $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_3 l_4 l_1 l_2}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} Q$.

By the above lemma and visibility, we can transform any transition of form $\Gamma \vdash_\phi P \stackrel{sl}{\Longrightarrow}$, with l output, to $\Gamma \vdash_\phi P \stackrel{tl}{\Longrightarrow}$ where $t = \ulcorner s \urcorner^{\mathsf{O}}$. Since an output is always unique (cf. Proposition 2 (ii)), we can now conclude:

Proposition 6. (innocence) *Let* $\Gamma \vdash_\psi P \stackrel{s_i l_i}{\Longrightarrow}$ $(i = 1, 2)$ *such that:* (1) *both sequences are legal;* (2) *both* l_1 *and* l_2 *are output; and* (3) $\ulcorner s_1 \urcorner^{\mathsf{O}} \equiv_\alpha \ulcorner s_2 \urcorner^{\mathsf{O}}$. *Then we have* $\ulcorner s_1 \urcorner^{\mathsf{O}} \cdot l_1 \equiv_\alpha \ulcorner s_2 \urcorner^{\mathsf{O}} \cdot l_2$.

Note that *contingency completeness* in [28] corresponds to the property that any legal trace ending in an output has a legal extension ending with an input, which is immediate by Proposition 3.2 (i) and typability of transitions. Therefore, up to redundant τ-actions, a sequential process is precisely characterised by the function mapping a set of output views to next actions. This is the *innocent function representation* of a sequential process.

It is now easy to see that well-knit legal traces of $\Gamma \vdash_\phi P_1|P_2$ are uniquely determined by those of $\Gamma \vdash_{\psi_i} P_i$ $(i = 1, 2)$ in the same way that innocent strategies are composed in the appropriate category of games [28].

4.5 Factoring Observables

An important property of $\cong_{\mathsf{seq}}$ is that any violation of $\cong_{\mathsf{seq}}$ can be detected by a tester process which is finite in the sense that the cardinality of the graph of its induced innocent function is finite. In particular, for our full abstraction result, we need finite processes which are type-wise translatable to (the interpretation of) PCF terms. To this end, we first show that the congruence $\cong_{\mathsf{seq}}$ can be obtained by only closing terms under $|$, given an appropriate base (*Context Lemma*, cf. [33]). Then we use the following result to unfold replication.

Proposition 7. (open replication) *Assume* $\Gamma \vdash_{\psi} P_1 \mid P_2 \mid R$ *where* R *is a replication with subject* x. *Then* $\Gamma \vdash_{\psi} P_1 \mid P_2 \mid R \cong_{\mathsf{seq}} (P_1 \mid R) \mid (\boldsymbol{\nu}\, x)(P_2 \mid R)$.

The proof of Proposition 7 uses a bisimulation induced by the typed transition (which stays within $\cong$). We can then establish the following proposition where $\overline{\Gamma}$ denotes the result of dualising each type occurring in Γ.

Proposition 8. (finite testability) *Assume* $\Gamma \vdash_{\mathtt{I}} P_i \triangleright \mathbf{?}_{\omega} y_1 \otimes \cdots \mathbf{?}_{\omega} y_n \otimes \mathbf{!}_{\omega} z$ $(i = 1, 2)$ *such that* $\mathsf{fn}(\Gamma) = \{\vec{y}, z\}$. *Then* $\Gamma \vdash_{\mathtt{I}} P_1 \ncong_{\mathsf{seq}} P_2$ *iff there exist finite* $\overline{\Gamma} \vdash_{\mathtt{I}} R_j \triangleright \mathbf{!}_{\omega} y_j$ $(1 \leq j \leq n)$ *and a finite* $\overline{\Gamma} \cdot x : \mathsf{Nat}^{\bullet} \vdash_{\mathtt{O}} S \triangleright \mathbf{?}_{\omega} z \otimes \mathbf{?}_1 x$ *such that* $(\Pi_j R_j | P_1 | S) \Downarrow_x$ *and* $(\Pi_j R_j | P_2 | S) \not\Downarrow_x$, *or its symmetric case.*

Towards the proof, we first take, by the Context Lemma mentioned above, a tester of form $\overline{\Gamma} \cdot x : \mathsf{Nat}^{\bullet} \vdash T'$ which, when composed with P_i, gives different observables. We then make, using Proposition 7, all shared replicated processes private to their "clients". This gives processes R'_i and S' which have the same types as R_i and S above. Finally, the shapes of types allow to consider processes R_i, P_i and S (to be precise by turning S to $!u(x).S$) as strategies in games. We can now appeal to finite testability in games, cf. [28], from which, by retranslating finite innocent strategies to finite processes, we conclude that finite testers suffice. Alternatively we can directly reason at the level of the π-calculus and its typed transitions, showing that any behaviour characterised by a finite innocent function (which is enough for testability) is realisable by (typable) syntactic processes [4, 45].

5 Full Abstraction

5.1 Interpretation

We consider PCF with a single base type, Nat, without loss of generality. Let $\alpha ::= \mathsf{Nat} \mid \alpha \Rightarrow \beta$. We write $[\alpha_1..\alpha_n \mathsf{Nat}]$ $(n \geq 0)$ for $\alpha_1 \Rightarrow (...(\alpha_n \Rightarrow \mathsf{Nat})..)$. Now the syntax of PCF terms are given by:

$$\begin{array}{rcl} M & ::= & x \mid \lambda x : \alpha.M \mid MN \mid n \mid \mathtt{succ}(M) \mid \mathtt{pred}(M) \\ & \mid & \mathtt{ifzero}\ M\ \mathtt{then}\ N\ \mathtt{else}\ L \mid \mu x : \alpha.M \end{array}$$

We omit operational semantics and the typing rules [15]. The mappings from PCF types and terms to π-types and terms, which are due to Hyland and Ong [29], are given in Figure 2. Copy-cat processes are given by $[x \to x']^{[\&_i \vec{\tau}_i]^{!_1}} \stackrel{\text{def}}{=} x[\&_i(\vec{y_i}).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}]$ and, for replicated types, $[x \to x']^{[\&_i \vec{\tau}_i]^{!_\omega}} \stackrel{\text{def}}{=} !x[\&_i(\vec{y_i}).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}]$. Copy-cats for unary types are special cases where the indexing sets are singletons. The interpretation of $[\alpha_1..\alpha_n \mathsf{Nat}]$ says a process, when asked for its value, asks back questions at types $\alpha_1, .., \alpha_n$, receives the results to these questions, and finally returns a natural number as the answer to the initial question.

(Type) $\mathsf{Nat}^\bullet \stackrel{\text{def}}{=} [\oplus_{i\in\mathbb{N}}]^{?_1}$ $\quad [\alpha_1..\alpha_{n-1}\mathsf{Nat}]^\circ \stackrel{\text{def}}{=} (\overline{\alpha_1^\circ}..\overline{\alpha_{n-1}^\circ}\mathsf{Nat}^\bullet)^{!_\omega}$

(Base) $\emptyset^\circ \stackrel{\text{def}}{=} \emptyset$ $\quad (E\cdot x:\alpha)^\circ \stackrel{\text{def}}{=} E^\circ \cdot x:\overline{\alpha^\circ}$

(Terms) Below we set $\beta = [\alpha_1..\alpha_{n-1}\mathsf{Nat}]$.

$[\![x:\alpha]\!]_u \stackrel{\text{def}}{=} [u \to x]^{\alpha^\circ}$

$[\![\lambda x_0 : \alpha_0.M : \alpha_0 \Rightarrow \beta]\!]_u \stackrel{\text{def}}{=} !\,u(x_0x_1..x_{n-1}z).(\boldsymbol{\nu}\,u')([\![M]\!]_{u'} \mid \mathsf{Arg}\langle u'x_1...x_{n-1}z\rangle^\beta)$

$[\![MN:\beta]\!]_u \stackrel{\text{def}}{=} !\,u(x_1..x_{n-1}z).(\boldsymbol{\nu}\,u'x_0)([\![M:\alpha\Rightarrow\beta]\!]_{u'} \mid [\![N:\alpha]\!]_{x_0} \mid \mathsf{Arg}\langle u'x_0...x_{n-1}z\rangle^{\alpha\Rightarrow\beta})$

$[\![\mathtt{n}:\mathsf{Nat}]\!]_u \stackrel{\text{def}}{=} !\,u(z).\overline{z}\mathtt{in}_n$

$[\![\mathtt{succ}(M):\mathsf{Nat}]\!]_u \stackrel{\text{def}}{=} !\,u(z).(\boldsymbol{\nu}\,x)([\![M]\!]_x \mid \overline{x}(y)y[\&_{n\in\mathbb{N}}\,\overline{z}\mathtt{in}_{n+1}])$

$[\![\mathtt{pred}(M):\mathsf{Nat}]\!]_u \stackrel{\text{def}}{=} !\,u(z).(\boldsymbol{\nu}\,x)([\![M]\!]_x \mid \overline{x}(y)y[\&_{n\in\mathbb{N}}\,\overline{z}\mathtt{in}_{n-1}])$

$[\![\mathtt{ifzero}\ M\ \mathtt{then}\ N\ \mathtt{else}\ L:\beta]\!]_u$

$\stackrel{\text{def}}{=} !\,u(x_1..x_{n-1}z).(\boldsymbol{\nu}\,m)([\![M]\!]_m \mid \overline{m}(z')z'[\&_i(\boldsymbol{\nu}\,u')(P_i \mid \mathsf{Arg}\langle u'x_1..x_{n-1}z\rangle^\beta)])$

where $P_0 \stackrel{\text{def}}{=} [\![N]\!]_{u'}$ else $P_i \stackrel{\text{def}}{=} [\![L]\!]_{u'}$.

$[\![\mu x:\alpha.M:\alpha]\!]_u \stackrel{\text{def}}{=} (\boldsymbol{\nu}\,m)([u\to m]^{\alpha^\circ} \mid [\![M:\alpha]\!]_m \mid [x\to m]^{\alpha^\circ})$

$\mathsf{Arg}\langle x\vec{y}z\rangle^{[\vec{\alpha}\mathsf{Nat}]} \stackrel{\text{def}}{=} \overline{x}(\vec{y}'z')(\Pi_i[y_i'\to y_i]^{\alpha_i^\circ} \mid [z'\to z]^{\overline{\mathsf{Nat}^\bullet}})$

Fig. 2. Encoding of PCF

5.2 Soundness

This is by the standard computational adequacy [33], which is proved by both-way operational correspondence, cf. [34]. Below let $\perp^{\mathsf{Nat}^\circ} \stackrel{\text{def}}{=} !\,u(z).\Omega_z^{\mathsf{Nat}^\circ}$ where Ω_u^τ is given in Example 3.5 (v).

Theorem 1. (computational adequacy) $M:\mathsf{Nat}\Downarrow$ *iff* $[\![M:\mathsf{Nat}]\!]_u \not\cong_{\mathsf{seq}} \perp^{\mathsf{Nat}^\circ}$.

Corollary 2. (soundness) $E \vdash M \cong N : \alpha$ *if* $E^\circ\cdot u:\alpha^\circ \vdash_{\mathtt{I}} [\![M:\alpha]\!]_u \cong_{\mathsf{seq}} [\![N:\alpha]\!]_u$

5.3 Completeness

Assume P is typed under (the interpretation of) a PCF-type and, moreover, it is finite, i.e. is representable by an innocent function. By [3, 28] or by a direct syntactic transformation, P can be mapped into a so-called *finite canonical-form*, which in turn is easily transformed to a standard PCF term without changing meaning in its interpretation up to $\cong_{\mathsf{seq}}$. Thus we obtain:

Theorem 2. (finite definability) *Let* $E^\circ\cdot u:\alpha^\circ \vdash_{\mathtt{I}} P \triangleright !_\omega u$ *be finite. Then* $E^\circ\cdot u:\alpha^\circ \vdash [\![M:\alpha]\!]_u \cong_{\mathsf{seq}} P$ *for some* M.

This result indicates that, in essence, *only sequential functional behaviour inhabits each type.* Now suppose $\vdash M_1 \cong M_2 : \alpha$ but $u:\mathsf{Nat}^\circ \vdash_{\mathtt{I}} [\![M_1]\!]_u \not\cong_{\mathsf{seq}} [\![M_2]\!]_u$.

Then the latter's difference is detectable by finite processes (Proposition 8). By Theorem 2 we can consider these finite testers as interpretations of PCF-terms so that we know, for example, $[\![C[M_1] : \mathsf{Nat}]\!] \Downarrow$ and $[\![C[M_2] : \mathsf{Nat}]\!] \Uparrow$. But this means, by Theorem 1, $C[M_1] : \mathsf{Nat} \Downarrow$ and $C[M_2] : \mathsf{Nat} \Uparrow$, contradicting our assumption. We have now reached the main result of the paper.

Theorem 3. (full abstraction) $E^\circ \cdot u : \alpha^\circ \vdash [\![M_1 : \alpha]\!]_u \cong_{\mathsf{seq}} [\![M_2 : \alpha]\!]_u$ *if and only if* $E \vdash M_1 \cong M_2 : \alpha$.

By replacing $\cong_{\mathsf{seq}}$ and $\cong$ with the corresponding precongruences, we similarly obtain inequational full abstraction.

5.4 Call-by-Value Sequentiality

This section briefly discusses how we can fully abstractly embed the call-by-value PCF to the same typed calculus by merely changing the interpretation of types. The call-by-value PCF uses the same set of types and terms except the standard restriction of recursion to terms of function types. We still write $\alpha, \beta, \ldots$ and $M, N, \ldots$ for types and preterms. The operational semantics is standard. The induced contextual congruence is denoted $\cong_v$.

The embeddings are given in Figure 3. The mapping of types is written $\alpha^\star$, defined using an auxiliary map denoted $\alpha^\diamond$. The mapping of typed terms takes the form $\langle\!\langle M : \alpha \rangle\!\rangle_u^{\mathcal{E}}$ where $\mathcal{E}$ is an environment which is a finite map from variables to natural numbers whose domain includes all free variables in M of type Nat. We write $\langle\!\langle M : \alpha \rangle\!\rangle_u$ when $\mathcal{E}$ is empty. We also use the following notations: (1) $\overline{x}\langle \vec{v} \rangle^{\vec{\tau}}$ stands for $\overline{x}(\vec{c})\Pi_i[c_i \to v_i]^{\tau_i}$, similarly for selection (cf. [18]); and (2) $C[\ ]_{i \in I}$ stands for a context with multiple holes indexed by I.

The encoding is directly based on the call-by-value game semantics introduced in [24]. As an example of the encoding of types, $\mathsf{Nat} \Rightarrow \mathsf{Nat}$ is interpreted as $([\&_{i \in \mathbb{N}}[\oplus_{i \in \mathbb{N}}]^{?_1}]^{!_\omega})^{?_1}$, which says that the function first signals itself, receives a natural number, then returns the result.

The notation $\overline{x}\langle \vec{y} \rangle$ would make the correspondence with Milner's call-by-value encoding in [35] clear (note the encoding in [35] only use a function type). The environment (albeit its slightly different format) has the same functionality as those used in the process encoding of call-by-value games presented in [12]. We can easily check $E \vdash M : \alpha$ (in the call-by-value PCF) implies $E^\diamond \cdot u : \alpha^\star \vdash_{\mathsf{0}} \langle\!\langle M : \alpha \rangle\!\rangle_u^{\mathcal{E}} \triangleright \mathbf{?}_\omega \vec{y} \otimes \mathbf{?}_1 u$ for any appropriate $\mathcal{E}$.

The proof of full abstraction follows that of the call-by-name PCF. First, using the same method as for Theorem 1, we can easily establish computational adequacy, hence soundness. For completeness, noting the results in §4.1 to §4.4 are given for general sequential processes, we immediately obtain:

Theorem 4. (finite definability) *Let* $E^\diamond \cdot u : \alpha^\star \vdash_{\mathsf{0}} P \triangleright \mathbf{?}_1 u$ *be finite where the range of* E *only contains function types. Then* $E^\diamond \cdot u : \alpha^\star \vdash \langle\!\langle M : \alpha \rangle\!\rangle_u \cong_{\mathsf{seq}} P$ *for some* $E \vdash M : \alpha$.

By Theorem 4 together with the analogue of Proposition 8 we finally obtain:

$$\textbf{(Type)}\quad \mathsf{Nat}^\star \stackrel{\text{def}}{=} [\oplus_{i\in\mathbb{N}}]^{?_1} \quad (\alpha\Rightarrow\beta)^\star \stackrel{\text{def}}{=} ((\alpha\Rightarrow\beta)^\diamond)^{?_1} \quad (\alpha\Rightarrow\beta)^\diamond \stackrel{\text{def}}{=} \begin{cases} [\&_{i\in\mathbb{N}}\beta^\star]^{!_\omega} & (\alpha=\mathsf{Nat}) \\ (\overline{\alpha^\diamond}\beta^\star)^{!_\omega} & (\text{else}) \end{cases}$$

$$\textbf{(Base)}\quad \emptyset^\diamond \stackrel{\text{def}}{=} \emptyset \quad (E\cdot x:\alpha)^\diamond \stackrel{\text{def}}{=} \begin{cases} E^\diamond & (\alpha=\mathsf{Nat}) \\ E^\diamond\cdot x:\overline{\alpha^\diamond} & (\alpha\neq\mathsf{Nat}) \end{cases}$$

(Terms)

$$\langle\!\langle x:\alpha\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} \begin{cases} \overline{u}\mathtt{in}_{\mathcal{E}(x)} & (\alpha=\mathsf{Nat}) \\ \overline{u}\langle c\rangle^{\alpha^\diamond} & (\text{else}) \end{cases}$$

$$\langle\!\langle \lambda x.M:\alpha\Rightarrow\beta\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} \begin{cases} \overline{u}(c)!c[\&_{n\in\mathbb{N}}(m).\langle\!\langle M:\beta\rangle\!\rangle^{\mathcal{E}[x\mapsto i]}_m & (\alpha=\mathsf{Nat}) \\ \overline{u}(c)!c(xm).\langle\!\langle M:\beta\rangle\!\rangle^{\mathcal{E}}_m & (\text{else}) \end{cases}$$

$$\langle\!\langle MN:\beta\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} \begin{cases} (\boldsymbol{\nu}\, mn)(\langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle^{\mathcal{E}}_m \mid m(c).(\langle\!\langle N:\alpha\rangle\!\rangle^{\mathcal{E}}_n \mid n[\&_{i\in\mathbb{N}}\overline{c}\mathtt{in}_i\langle u\rangle^{\beta^\star}])) & (\alpha=\mathsf{Nat}) \\ (\boldsymbol{\nu}\, mn)(\langle\!\langle M:\alpha\Rightarrow\beta\rangle\!\rangle^{\mathcal{E}}_m \mid m(c).(\langle\!\langle N:\alpha\rangle\!\rangle^{\mathcal{E}}_n \mid n(e).\overline{c}\langle eu\rangle^{\alpha^\diamond\beta^\star})) & (\text{else}) \end{cases}$$

$$\langle\!\langle n:\mathsf{Nat}\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} \overline{u}\mathtt{in}_n$$

$$\langle\!\langle \mathtt{succ}(M):\mathsf{Nat}\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, m)(\langle\!\langle M:\mathsf{Nat}\rangle\!\rangle^{\mathcal{E}}_m | m[\&_{i\in\mathbb{N}}\overline{u}\mathtt{in}_{i+1}])$$

$$\langle\!\langle \mathtt{ifzero}\ M\ \mathtt{then}\ N\ \mathtt{else}\ L:\beta\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, m)(\langle\!\langle M:\mathsf{Nat}\rangle\!\rangle^{\mathcal{E}}_m | m[\&_{i\in\mathbb{N}}P_i]) \quad (\text{where } P_0 \stackrel{\text{def}}{=} \langle\!\langle N:\beta\rangle\!\rangle^{\mathcal{E}}_u,\ P_{i+1} \stackrel{\text{def}}{=} \langle\!\langle L:\beta\rangle\!\rangle^{\mathcal{E}}_u)$$

$$\langle\!\langle \mu x.M:\alpha\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} C[\overline{u}(c)(\boldsymbol{\nu}\, x)(P_i|[x\to c]^{\alpha^\diamond})]_{i\in I} \qquad (\text{where } \langle\!\langle M:\alpha\rangle\!\rangle^{\mathcal{E}}_u \stackrel{\text{def}}{=} C[\overline{u}(c)P_i]_{i\in I})$$

Fig. 3. Encoding of Call-by-Value PCF

Theorem 5. (full abstraction) $E^\diamond\cdot u:\alpha^\star \vdash \langle\!\langle M_1:\alpha\rangle\!\rangle_u \cong_{\mathsf{seq}} \langle\!\langle M_2:\alpha\rangle\!\rangle_u$ *if and only if* $E\vdash M_1\cong_v M_2:\alpha$.

References

1. Abramsky, S., Computational interpretation of linear logic. *TCS*, Vol. 111, 1993.
2. Abramsky, S., Honda, K. and McCusker, G., A Fully Abstract Game Semantics for General References. *LICS*, 334-344, IEEE, 1998.
3. Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF. *Info. & Comp.*, Vol. 163, 2000.
4. Berger, M. Honda, K. and N. Yoshida. *Genericity in the π-Calculus.* To appear as a QMW DCS Technical Report, 2001.
5. Berry, G. and Curien, P. L., Sequential algorithms on concrete data structures *TCS*, 20(3), 265-321, North-Holland, 1982.
6. Boreale, M. and Sangiorgi, D., Some congruence properties for π-calculus bisimilarities, *TCS*, 198, 159–176, 1998.
7. Boudol, G., Asynchrony and the pi-calculus, INRIA Research Report 1702, 1992.
8. Boudol, G., The pi-calculus in direct style, *POPL'97*, 228–241, ACM, 1997.
9. Cartwright, R., Curien, P.-L. and Felleisen, M., Full abstract semantics for observably sequential languages. *I & C*, 1994.

10. Curien, P. L., Sequentiality and full abstraction. Proc. of *Application of Categories in Computer Science*, LNM 177, 86–94, Cambridge Press, 1995.
11. Degano, P. and Priami, C. Proved Trees, *ICALP'92*, LNCS 623, 629–640, Springer 1992.
12. Fiore, M. and Honda, K., Recursive Types in Games: axiomatics and process representation, *LICS'98*, 345-356, IEEE, 1998.
13. Gay, S. and Hole, M., Types and Subtypes for Client-Server Interactions, *ESOP'99*, LNCS 1576, 74–90, Springer, 1999.
14. Girard, J.-Y., Linear Logic, *TCS*, Vol. 50, 1–102, 1987.
15. Gunter, C., *Semantics of Programming Languages: Structures and Techniques*, MIT Press, 1992.
16. Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, 509-523, 1993.
17. Honda, K., Composing Processes, *POPL'96*, 344-357, ACM, 1996.
18. Honda, K., Notes on Linear Typing for Free Outputs, May, 2001. Available at www.dcs.qmw.ac.uk/ kohei.
19. Honda, K., Notes on the linear π-calculus and LLP, June, 2001. Available at www.dcs.qmw.ac.uk/ kohei.
20. Honda, K., Kubo, M. and Vasconcelos, V., Language Primitives and Type Discipline for Structured Communication-Based Programming. *ESOP'98*, LNCS 1381, 122–138. Springer-Verlag, 1998.
21. Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, 133–147, Springer-Verlag 1991.
22. Honda, K. Vasconcelos, V., and Yoshida, N. Secure Information Flow as Typed Process Behaviour, *ESOP '99*, LNCS 1782, 180–199, Springer-Verlag, 2000.
23. Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *TCS*, 437–486, Vol. 151, North-Holland, 1995.
24. Honda, K. and Yoshida, N. Game-theoretic analysis of call-by-value computation. *TCS* Vol. 221 (1999), 393–456, North-Holland, 1999.
25. Honda, K. and Yoshida, N. A Uniform Type Structure for Secure Information Flow, July 2001, available at www.mcs.le.ac.uk/ nyoshida/paper.html.
26. Kobayashi, N., A partially deadlock-free typed process calculus, *ACM TOPLAS*, Vol. 20, No. 2, 436–482, 1998.
27. Kobayashi, N., Pierce, B., and Turner, D., Linear Types and π-calculus, *POPL'96*, 358–371, ACM Press, 1996.
28. Hyland, M. and Ong, L., On Full Abstraction for PCF: I, II and III. 130 pages, 1994. To appear in *Info. & Comp.*
29. Hyland, M. and Ong, L., Pi-calculus, dialogue games and PCF, *FPCA'95*, ACM, 1995.
30. Laird, J., Full abstraction for functional languages with control, *LICS'97*, IEEE, 1997.
31. Longley, J., The sequentially realizable functionals, Annals of Pure and Applied Logic, Available as LFCS Report ECS-LFCS-98-402. December 1998.
32. McCusker, G., Games and Full Abstraction for FPC. *LICS'96*, IEEE, 1996.
33. Milner, R., Fully abstract models of typed lambda calculi. *TCS*, 4:1–22, 1977.
34. Milner, R., Functions as Processes. *MSCS*, 2(2), 119–146, CUP, 1992.
35. Milner, R., Polyadic π-Calculus: a tutorial. *Proceedings of the International Summer School on Logic Algebra of Specification*, Marktoberdorf, 1992.
36. Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), 1–77, 1992.
37. Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *LICS'93*, 187–215, IEEE, 1993.

38. Quaglia, P. and Walker, D., On Synchronous and Asynchronous Mobile Processes, *FoSSaCS 00*, LNCS 1784, 283–296, Springer, 2000.
39. Sangiorgi, D., *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms.* Ph.D. Thesis, University of Edinburgh, 1992.
40. Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, North-Holland, 1996.
41. Sangiorgi, D., The name discipline of uniform receptiveness, *ICALP'97*, LNCS 1256, 303–313, Springer, 1997.
42. Vasconcelos, V., Typed concurrent objects. *ECOOP'94*, LNCS 821, 100–117. Springer, 1994.
43. Vasconcelos, V. and Honda, K., Principal Typing Scheme for Polyadic π-Calculus. *CONCUR'93*, LNCS 715, 524–538, Springer-Verlag, 1993.
44. Yoshida, N., Graph Types for Monadic Mobile Processes, *FST/TCS'16*, LNCS 1180, 371–387, Springer-Verlag, December, 1996. Full version as LFCS Technical Report, ECS-LFCS-96-350, 1996.
45. N. Yoshida., Berger, M. and Honda, K., Strong Normalisation in the π-Calculus, *LICS'01*, IEEE, 2001. Full version as MCS Technical Report, 2001-09, May, 2001.

Summary of Appendices A–J

Appendices offer details of definitions and technical development omitted in the main sections. In particular they contain detailed proofs of the central results of the paper. The appendices are organised as follows.

- **A** Typing Rules for Branching.
- **B** Definition of copy-cats.
- **C** Definitions of structural congruence, reduction relation and transition relation for the untyped processes.
- **D** Properties of the typing system as well as a couple of the properties of $\cong_{\mathsf{seq}}$ (Section 3).
- **E** Formal definitions and basic properties of typed transitions (Section 4).
- **F** Formal treatment of occurrences and transitions enriched with redexes.
- **G** Proofs of the results stated in Sections 4.3 and 4.4 (visibility, well-bracketing and innocence).
- **H** The proof of Context Lemma (Section 4.5).
- **I** The proof of Finite Tester Lemma (Section 4.6).
- **J** The proof of Computational Adequacy and Finite Definability (Section 5).

Throughout the rest of the appendix we assume the standard variable convention for bound names.

A Typing Rules for Branching

$$\frac{(\mathsf{Bra}^{!_1})\quad (C_i/\vec{y}_i = \mathbf{?}A)\qquad \Gamma \vdash x : [\&_{i\in I}\vec{\tau}_i]^{!_1}\qquad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash_{\mathtt{O}} P_i \triangleright C_i^{-x}}{\Gamma \vdash_{\mathtt{I}} x[\&_{i\in I}(\vec{y}_i : \vec{\tau}_i).P_i] \triangleright A \otimes \mathbf{!}_1 x}$$

$$\frac{(\mathsf{Sel}^{?_1})\quad (C_i/\vec{y}_i = A \asymp \mathbf{?}_1 x)\qquad \Gamma \vdash x : [\oplus_{i\in I}\vec{\tau}_i]^{?_1}\qquad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash_{\mathtt{I}} P \triangleright C}{\Gamma \vdash_{\mathtt{O}} \overline{x}\mathtt{in}(\vec{y}_i : \vec{\tau}_i)P \triangleright A \odot \mathbf{?}_1 x}$$

$$\frac{(\mathsf{Bra}^{!_\omega})\quad (C_i/\vec{y}_i = \mathbf{?}_\omega A)\qquad \Gamma \vdash x : [\&_{i\in I}\vec{\tau}_i]^{!_1}\qquad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash_{\mathtt{O}} P_i \triangleright C_i^{-x}}{\Gamma \vdash_{\mathtt{I}} !x[\&_{i\in I}(\vec{y}_i : \vec{\tau}_i).P_i] \triangleright A \otimes \mathbf{!}_\omega x}$$

$$\frac{(\mathsf{Sel}^{?_\omega})\quad (C_i/\vec{y}_i = A \asymp \mathbf{?}_\omega x)\qquad \Gamma \vdash x : [\oplus_{i\in I}\vec{\tau}_i]^{?_\omega}\qquad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash_{\mathtt{I}} P \triangleright C}{\Gamma \vdash_{\mathtt{O}} \overline{x}\mathtt{in}(\vec{y}_i : \vec{\tau}_i)P \triangleright A \odot \mathbf{?}_1 x}$$

B Copy-cat Processes

The copy cat $[x \to y]^{\tau_I}$ is given by:

$$[x \to x']^{(\vec{\tau})^{!_1}} \stackrel{\mathrm{def}}{=} x(\vec{y}).\overline{x'}(\vec{y'})\Pi_i[y'_i \to y_i]^{\overline{\tau_i}}$$

$$[x \to x']^{[\&_i\vec{\tau}_i]^{!_1}} \stackrel{\mathrm{def}}{=} x[\&_i(\vec{y}_i).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}$$

$$[x \to x']^{(\vec{\tau})^{!_\omega}} \stackrel{\mathrm{def}}{=} !x(\vec{y}).\overline{x'}(\vec{y'})\Pi_i[y'_i \to y_i]^{\overline{\tau_i}}$$

$$[x \to x']^{[\&_i\vec{\tau}_i]^{!_\omega}} \stackrel{\mathrm{def}}{=} !x[\&_i(\vec{y}_i).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}$$

C Reduction and Transition

C.1 Structural Congruence

The *structural congruence* $\equiv$ is the least congruence generated by:

- If $P \equiv_\alpha Q$ then $P \equiv Q$.
- $P|\mathbf{0} \equiv P$, $P|Q \equiv Q|P$ and $(P|Q)|R \equiv P|(Q|R)$.
- $(\boldsymbol{\nu}\, x)\mathbf{0} \equiv \mathbf{0}$ and $(\boldsymbol{\nu}\, x)(\boldsymbol{\nu}\, y)P \equiv (\boldsymbol{\nu}\, y)(\boldsymbol{\nu}\, x)P$.
- If $x \notin \mathsf{fn}(P)$ then $(\boldsymbol{\nu}\, x)(P|Q) \equiv P|(\boldsymbol{\nu}\, x)Q$.
- If $\{\vec{y}\}\cap\mathsf{fn}(P) = \emptyset$ then $\overline{x}(\vec{y})(P|Q) \equiv P|\overline{x}(\vec{y})Q$ and $\overline{x}\mathtt{in}_i(\vec{y})(P|Q) \equiv P|\overline{x}\mathtt{in}_i(\vec{y})Q$.
- If $x \notin \{y, \vec{z}\}$, then $(\boldsymbol{\nu}\, x)\overline{y}(\vec{z})P \equiv \overline{y}(\vec{z})(\boldsymbol{\nu}\, x)P$ and $(\boldsymbol{\nu}\, x)\overline{y}\mathtt{in}_i(\vec{z})P \equiv \overline{y}\mathtt{in}_i(\vec{z})(\boldsymbol{\nu}\, x)P$.

The axioms given under the last two clauses are not usually among those use for generating structural congruences for π-calculi.

C.2 Reduction Rules

The relation $\longrightarrow$ on untyped terms is generated by the following rules.

$$
\begin{array}{rl}
(\textsc{Com}) & x(\vec{y}).P \mid \overline{x}(\vec{y})Q \longrightarrow (\boldsymbol{\nu}\, \vec{y})(P|Q) \\
(\textsc{Bra}) & x[\&_i(\vec{y}_i).P_i] \mid \overline{x}\mathtt{in}_i(\vec{y}_i)Q \longrightarrow (\boldsymbol{\nu}\, \vec{y}_i)(P_i \mid Q) \\
(\textsc{Com}_!) & !\,x(\vec{y}).P \mid \overline{x}(\vec{y})Q \longrightarrow !\,x(\vec{y}).P|(\boldsymbol{\nu}\, \vec{y})(P|Q) \\
(\textsc{Bra}_!) & !\,x[\&_i(\vec{y}_i).P_i] \mid \overline{x}\mathtt{in}_i(\vec{y}_i)Q \longrightarrow !\,x[\&_i(\vec{y}_i).P_i]|(\boldsymbol{\nu}\, \vec{y}_i)(P_i|Q) \\
(\textsc{Res}) & P \longrightarrow Q \implies (\boldsymbol{\nu}\, x)P \longrightarrow (\boldsymbol{\nu}\, x)Q \\
(\textsc{Par}) & P \longrightarrow P' \implies P|Q \longrightarrow P'|Q \\
(\textsc{Out}) & P \longrightarrow Q \implies \overline{x}(\vec{y})P \longrightarrow \overline{x}(\vec{y})Q \\
(\textsc{Bout}) & P \longrightarrow Q \implies \overline{x}\mathtt{in}_i(\vec{y})P \longrightarrow \overline{x}\mathtt{in}_i(\vec{y})Q \\
(\textsc{Cong}) & P \equiv P' \longrightarrow Q' \equiv Q \implies P \longrightarrow Q
\end{array}
$$

C.3 Transitions

The *untyped* transition relation $\xrightarrow{l}$ has labels of the form τ, $x(\vec{y})$, $\overline{x}(\vec{y})$, $x\mathtt{in}_i(\vec{y})$ and $\overline{x}\mathtt{in}_i(\vec{y})$ and is inductively generated by the following rules.

$$
\begin{array}{rlrl}
(\textsc{In}) & x(\vec{y}).P \xrightarrow{x(\vec{y})} P & (\textsc{Bra}) & x[\&_{i\in I}(\vec{y}_i).P_i] \xrightarrow{x\mathtt{in}_j(\vec{y}_j)} P_j \\
(\textsc{In}_!) & !\,x(\vec{y}).P \xrightarrow{x(\vec{y})} !x(\vec{y}).P|P & (\textsc{Bra}_!) & !\,x[\&_{i\in I}(\vec{y}_i).P_i] \xrightarrow{x\mathtt{in}_j(\vec{y}_j)} !x[\&_{i\in I}(\vec{y}_i).P_i]|P_j \\
(\textsc{Out}) & \overline{x}(\vec{z})P \xrightarrow{\overline{x}(\vec{z})} P & (\textsc{Sel}) & \overline{x}\mathtt{in}_i(\vec{z})P \xrightarrow{\overline{x}\mathtt{in}_i(\vec{z})} P
\end{array}
$$

$$\text{(COM)} \quad P \xrightarrow{x(\vec{y})} P', Q \xrightarrow{\overline{x}(\vec{y})} Q' \implies P|Q \xrightarrow{\tau} (\nu\vec{y})(P'|Q')$$

$$\text{(COM}_i\text{)} \quad P \xrightarrow{x\mathtt{in}_i(\vec{y})} P', Q \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} Q' \implies P|Q \xrightarrow{\tau} (\nu\vec{y})(P'|Q')$$

$$\text{(RES)} \quad P \xrightarrow{l} Q, x \notin \mathsf{sbj}(l) \implies (\nu x)P \xrightarrow{l} (\nu x)Q$$

$$\text{(PAR)} \quad P \xrightarrow{l} P', \mathsf{fn}(Q) \cap \mathsf{bn}(l) = \emptyset \implies P|Q \xrightarrow{l} P'|Q$$

$$\text{(OUT-}\xi\text{)} \quad P \xrightarrow{l} P', \mathsf{fn}(l) \cap \{\vec{y}\} = \emptyset \implies \overline{x}(\vec{y})P \xrightarrow{l} \overline{x}(\vec{y})P'$$

$$\text{(OUT-}\xi_i\text{)} \quad P \xrightarrow{l} P', \mathsf{fn}(l) \cap \{\vec{y_i}\} = \emptyset \implies \overline{x}\mathtt{in}_i(\vec{y_i})P \xrightarrow{l} \overline{x}\mathtt{in}_i(\vec{y_i})P'$$

$$\text{(ALPHA)} \quad P \equiv_\alpha P' \xrightarrow{l} Q' \equiv_\alpha Q \implies P \xrightarrow{l} Q$$

Theorem 6. *Reductions coincide with τ-transitions: $P \longrightarrow Q$ if and only if $P \xrightarrow{\tau}\equiv Q$.*

PROOF: Standard, along the lines of [34]. ■

In the rest of this text, the notation $P \longrightarrow Q$ signifies that the processes P, Q *may* be untyped while $\Gamma \vdash_\phi P \longrightarrow Q$ indicates that P and Q are sequentially typed, and similarly for transitions.

D Proofs for Section 3

D.1 Properties of the typing system

Most of the proofs of Section 3, which establish basic properties of the sequential typing system, are the same as those in the long version of [22], except that the proofs are simpler since the present typing system has no ordering between nodes and secrecy levels,

Proof of Proposition 1 **(i)** By induction on the derivation of typing rules in Figure 1. First we note that if τ is sequential, then $\langle\tau,\ \overline{\tau}\rangle$ is sequential. For (`Zero`), suppose $\Delta \leq \Gamma$ and $\Delta \vdash_{\mathtt{I}} \mathbf{0}$. Then, if Δ is sequential, Γ must also be sequential. Hence we have $\Gamma \vdash_{\mathtt{I}} \mathbf{0}$. Similarly for other cases, noting that for all x such that $\Delta \vdash x\!:\!\tau$, if $\Delta \leq \Gamma$, then we have $\Gamma \vdash x\!:\!\tau$.

(ii) For the former, we can construct the minimum typing system by deleting weakening rules as in Appendix C in [22]. The latter is immediate from (i).

(iii) We first prove that $\Gamma \vdash_\phi P \triangleright A$ and $P \equiv Q$ imply $\Gamma \vdash_\phi Q \triangleright A$.

First, associativity $P \mid Q \equiv Q \mid P$ is immediate by definition.

For $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, we have to prove commutativity of IO-modes $(\phi_1 \odot \phi_2) \odot \phi_3 = \phi_1 \odot (\phi_2 \odot \phi_3)$ and commutativity of action types $(A \odot B) \odot C = A \odot (B \odot C)$. Both are proved by case analysis. For IO-modes, if $(\phi_1 \odot \phi_2) \odot \phi_3$ is defined, then we have either (1) all three modes are $\mathtt{I}$ or (2) one of three is $\mathtt{O}$ and others are $\mathtt{I}$. Both cases are proved mechanically. For action types, we only have to check commutativity of modes of each action node, $p \odot (q \odot r) = (p \odot q) \odot r$. Then, if $p \odot (q \odot r)$ is defined, either (1) all three modes are $?_\omega$ or (2) one of the three is $!_\omega$ and the others are $?_\omega$. Hence the proof is again mechanical.

The scope opening rule $(\boldsymbol{\nu}\, a)(P \mid Q) \equiv (\boldsymbol{\nu}\, a)P \mid Q$ is the same as the corresponding proof in [22] because this is not related to IO-modes. The rule $(\boldsymbol{\nu}\, z)\overline{x}(\vec{y})\cdot P \equiv \overline{x}(\vec{y})\cdot(\boldsymbol{\nu}\, z)P$ is also similar to the rule $(\boldsymbol{\nu}\, z)(\boldsymbol{\nu}\, y)P \equiv (\boldsymbol{\nu}\, y)(\boldsymbol{\nu}\, z)P$.

The only remaining interesting rule is $\overline{x}(\vec{z}:\vec{\tau})\cdot(P|Q) \equiv (\overline{x}(\vec{z}:\vec{\tau})\cdot P)|Q$ with $\mathsf{fn}(Q) \cap \{\vec{z}\} = \emptyset$. Here we only have to check IO-modes (hence below we omit action types). Suppose $\Gamma \vdash_\phi \overline{x}(\vec{z})\cdot(P|Q)$. Then $\phi = \mathtt{o}$ and $\Gamma\cdot\vec{z}:\vec{\tau} \vdash_{\mathtt{I}} (P|Q)$. If $\mathtt{I} = \phi \odot \psi$, then $\phi = \psi = \mathtt{I}$. So the sequent must be derived from $\Gamma\cdot\vec{z}:\vec{\tau} \vdash_{\mathtt{I}} P$ and $\Gamma\cdot\vec{z}:\vec{\tau} \vdash_{\mathtt{I}} Q$. Then by applying $(\mathtt{In}^{!_1})$ or $(\mathtt{In}^{!_\omega})$, we have $\Gamma \vdash_{\mathtt{o}} \overline{x}(\vec{z})\cdot P$. By $\mathsf{fn}(Q) \cap \{\vec{z}\} = \emptyset$, we have $\Gamma \vdash_{\mathtt{I}} Q$. Hence by $\mathtt{o} \odot \mathtt{I} = \mathtt{o}$, we have $\Gamma \vdash_{\mathtt{I}} (\overline{x}(\vec{z}:\vec{\tau})\cdot P)|Q$. The other direction is similar.

Now we prove that $\Gamma \vdash_\phi P \triangleright A$ and $P \longrightarrow Q$ imply $\Gamma \vdash_\phi Q \triangleright A$ by induction on the derivation of $\longrightarrow$. Suppose that $\Gamma \vdash_\phi x(\vec{y}:\vec{\tau}_1).P \,|\, \overline{x}(\vec{y}:\vec{\tau}_2)\cdot Q \triangleright A$ and

$$x(\vec{y}:\vec{\tau}_1).P \,|\, \overline{x}(\vec{y}:\vec{\tau}_2)\cdot Q \;\longrightarrow\; (\boldsymbol{\nu}\, \vec{y}:\langle \vec{\tau}_1,\ \vec{\tau}_2\rangle)(P \,|\, Q).$$

Then we have $\Gamma \vdash_{\mathtt{I}} x(\vec{y}:\vec{\tau}_1).P \triangleright\, !_1 x \odot A_1$ and $\Gamma \vdash_{\mathtt{o}} \overline{x}(\vec{y}:\vec{\tau}_2)\cdot Q \triangleright\, ?_1 x \odot A_2$ with $A = \bot x \otimes A_1 \odot A_2$ and $\Gamma \vdash x : !_1, ?_1$. Note that $\phi = \mathtt{o}$ by $\mathtt{o} \odot \mathtt{I} = \mathtt{o}$. The above two sequences are derived from

$$\Gamma\cdot\vec{y}:\vec{\tau}_1 \vdash_{\mathtt{o}} P \triangleright A_1 \quad \text{and} \quad \Gamma\cdot\vec{y}:\vec{\tau}_2 \vdash_{\mathtt{I}} Q \triangleright A_2$$

Then by Proposition 1 (i), we know $\Gamma\cdot\vec{y}:\langle \vec{\tau}_1,\ \vec{\tau}_2\rangle \vdash_{\mathtt{o}} P \triangleright A_1$ and $\Gamma\cdot\vec{y}:\langle \vec{\tau}_1,\ \vec{\tau}_2\rangle \vdash_{\mathtt{I}} Q \triangleright A_2$. Hence we have

$$\Gamma\cdot\vec{y}:\langle \vec{\tau}_1,\ \vec{\tau}_2\rangle \vdash_{\mathtt{o}} P \mid Q \triangleright A_1 \odot A_2$$

Note that $x \notin \mathsf{fn}(A_1 \odot A_2)$ and $\Gamma \vdash x : !_1, ?_1$. Hence by using $(\mathsf{Weak}\text{-}\bot)$ and then applying (Res), we can conclude the proof. The input replication case is similar to this case, the only difference being modes of action types, cf. [22]. Other cases, including the rule $P \longrightarrow Q \;\Rightarrow\; \overline{x}(\vec{z})P \longrightarrow \overline{x}(\vec{z})Q$, are straightforward from the induction hypothesis. ■

Proof of Proposition 2 **(i)** is obvious by the side condition $?A$ and $?_\omega A$ in the premises of $(\mathsf{In}^{!_1})$ and $(\mathsf{In}^{!_\omega})$, respectively. **(ii)** is also straightforward: for (1), if there is an active output subject, we can write $P \equiv \overline{x}(\vec{z})P'$. Since typing rules are closed under $\equiv$, we have $\Gamma \vdash_{\mathtt{o}} P$, which contradicts the assumption. (2) and (3) are obvious since neither $\mathtt{o} \odot \mathtt{o}$, $!_\omega \odot !_\omega$ nor $!_1 \odot !_1$ is defined. ■

Proof of Corollary 1 By Proposition 2 (ii)(2,3), we always have at most one active input and at most one active output for each free name; hence $Q_1 \equiv Q_2$. Furthermore, $\phi = \mathtt{o}$ follows by (ii)(1). ■

We shall also use the following small fact later.

Fact 1 *Let $\overline{x}(\vec{y})(P|Q)$ be typable and Q's active subjects are not among $\vec{y}$. Then* $\mathsf{fn}(Q) \cap \{\vec{y}\} = \emptyset$.

PROOF: This is because an output only exports input channels which should always be among active subjects of the body. ■

Fact 1 indicates that, in output prefixing rules, we can assume all active subjects (i.e. free subjects which are not prefixed under input prefixes) of a prefixed body are to be abstracted. Thus we can use the following restricted output prefixing rules instead of the original ones (showing only the unary cases and writing $\mathsf{Act}(P)$ for the active subjects of P).

$$\frac{(\mathsf{Out}^{?_1}) \quad (C/\vec{y} = A \asymp ?_1 x) \quad \Gamma \vdash x : (\vec{\tau})^{?_1} \quad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P \triangleright C \quad \mathsf{Act}(P) = \{\vec{y}\}}{\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y} : \vec{\tau})P \triangleright A \odot ?_1 x} \qquad \frac{(\mathsf{Out}^{?_\omega}) \quad (C/\vec{y} = A \asymp ?_\omega x) \quad \Gamma \vdash x : (\vec{\tau})^{?_\omega} \quad \Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P \triangleright C \quad \mathsf{Act}(P) = \{\vec{y}\}}{\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y} : \vec{\tau})P \triangleright A \odot ?_\omega x} \tag{8}$$

Because the premise should be in I-mode, we are automatically restricting P above to be parallel composition of input processes whose subjects are $\vec{y}$. Fact 1 ensures we can retain the same typability up to $\equiv$. This reduced typing system simplifies induction on the derivation of typing judgements since reduction and transitions do not rely on transformation of terms by $\equiv$.

D.2 Contextual Equality and Reduction

In Section 3.7, we introduced the basic typed equality on sequential processes $\cong_{\mathsf{seq}}$. In the following we offer the proof that reduction is contained in this relation, which also illuminates the dynamics of sequential processes. We use the strong barbs (observables) defined as follows. Below, as in Section 3.7, we say x is *active* when it is the free subject of an active input or output.

$$\Gamma \vdash_{\mathtt{O}} P \downarrow_x \quad \overset{\text{def}}{\Leftrightarrow} \quad x \text{ is active in } P' \text{ and } \Gamma \vdash_{\mathtt{O}} P \triangleright A \otimes ?_1 x.$$

Lemma 2. *If $\Gamma \vdash_\phi P|Q \downarrow_x$ then one of the following must be the case: (1) $P \downarrow_x$ and $Q \not\downarrow_x$ or (2) $P \not\downarrow_x$ and $Q \downarrow_x$.*

PROOF: By straightforward induction on derivation of the typing judgement. ■

Lemma 3. *If $\Gamma \vdash_\phi P \longrightarrow Q$, then $\Gamma \vdash_\phi P \not\downarrow_x$.*

PROOF: Assume that $P \longrightarrow Q$. By Corollary 1 this means $\phi = \mathtt{o}$, hence by Proposition 2 (ii) there is a unique active output, say P_0, in P, hence $P \equiv (\boldsymbol{\nu}\, \vec{z})(P_0|Q_0|R)$ such that Q_0 is an input with subject x and $x \notin \{\vec{z}\}$. Now assume further $\Gamma \vdash_{\mathtt{O}} P \downarrow_x$. By definition there is an active output with free subject x in P which therefore should coincide with P_0. Hence Q_0 above is an input with free subject x. Then by easy rule induction we have $\Gamma \vdash_{\mathbf{0}} (\boldsymbol{\nu}\, \vec{z})(P_0|Q_0|R) \triangleright B$ such that $px \in B$ with either $p = \mathbf{!}_\omega$ or $p = \bot$, which contradicts $\Gamma \vdash_\phi P \downarrow_x$, as required. ■

Lemma 4. *If $C[P]$ is typable and $C[P] \downarrow_x$ then either $C[\,] \downarrow_x$ or $P \downarrow_x$.*

PROOF: By straightforward induction on the structure of $C[\,]$. ■

Below and henceforth we write $C[\,] \downarrow_x$ and $C[\,] \longrightarrow C'[\,]$ when the processes which constitute the given context has a barb and a reduction by itself, respectively, in the latter case preserving the binding over the hole.

Lemma 5. *Assume that $P \longrightarrow Q$ and $C[\,]$ is a context (which might not be a reduction context) such that $C[P]$ is typable. Then $C[P] \longrightarrow R$ implies that exactly one of the following is the case: (1) $C[\,]$ is not a reduction-context and $R = C'[P]$ and $C[\,] \longrightarrow C'[\,]$; and (2) $C[\,]$ is a reduction context and $R = C[Q]$.*

PROOF: The case when $C[\,]$ is not a reduction-context is immediate. Suppose $C[\,]$ is a reduction context. Since $P \longrightarrow Q$, there is an active output together with a matching active input in P which is still active in $C[P]$ as C is a reduction context. Since $C[P]$ is typable, $C[\,]$ itself cannot contain an active output. But $C[\,]$ cannot contain a matching input redex either since the subject for an input can only occur once as input in any typed term. Hence $C[\,]$ can neither contribute an input nor an output to a reduction of $C[P]$ and (2) is the only possibility for $C[P]$'s reductions. ■

Lemma 6. *If $\Gamma \vdash_\phi P \longrightarrow Q$ then $\Gamma \vdash_\phi P \cong_{\text{seq}} Q$.*

PROOF: Choose an appropriate context $C[\,]$ and a weak barb $\Downarrow_x$. Clearly $C[Q] \Downarrow_x$ implies $C[P] \Downarrow_x$. For the reverse implication, assume $\Delta \vdash_\psi C[P] \Downarrow_x$ and let n be such that $C[P] = R_0 \longrightarrow \cdots \longrightarrow R_n \downarrow_x$ (n is determined uniquely by determinacy). By partial induction on $i \in \{0, \ldots, n\}$ we show that one of the following must be the case for R_i.

1. $R_i = C'[P]$ where $C[\,] \longrightarrow C'_1[\,] \longrightarrow \ldots \longrightarrow C'_i[\,] \stackrel{\text{def}}{=} C'[\,]$ and, moreover, for $0 \le j \le i-1$, $C'_j[\,]$ is not a reduction context.
2. $C[Q] \twoheadrightarrow R_i$.

In the base case, (1) holds trivially. For the inductive step, assume that $0 < i < n$. If (2) holds for R_i, then it also holds for R_{i+1}, so assume that (1) is the case. If C'_i is not a reduction context we immediately have (1). By Lemma 5 exactly one of the following must be true:

- $R_i = C'[P] \longrightarrow R_{i+1}$ is $C'[P] \longrightarrow C''[P]$ for some reduction context $C''[\,]$ such that $C'[\,] \longrightarrow C''[\,]$, or
- $R_i = C'[P] \longrightarrow C'[Q] = R_{i+1}$.

If the former is the case, then (1) holds for $i+1$. Otherwise, $C[\ ] \twoheadrightarrow C'[\ ]$ and $C[Q] \longrightarrow C'[Q] = R_{i+1}$ imply (2). This concludes the induction.

Now assume that (1) holds for R_n: then Lemma 4 allows for only the following two mutually exclusive possibilities: $C'[\ ] \downarrow_x$ and $P \downarrow_x$. The latter is in violation of Lemma 3, so we are left with the former, hence $C[Q] \downarrow_x$. On the other hand if (2) holds then $C[Q] \Downarrow_x$ is immediate. ∎

Theorem 7. *If $P \twoheadrightarrow Q$ then $P \cong_{\mathsf{seq}} Q$.*

PROOF: By $\equiv \subseteq \cong_{\mathsf{seq}}$ and $\longrightarrow \circ \longrightarrow \subseteq \cong_{\mathsf{seq}} \circ \cong_{\mathsf{seq}} = \cong_{\mathsf{seq}}$. ∎

D.3 Contextual Equality as a Maximal Consistent Theory

cAs we briefly noted in Section 3.7, $\cong_{\mathsf{seq}}$ arises as a certain maximal consistent equality. We outline the essential part of the proof in the following. Let $\cong$ be a sound equality in the sense of [23], i.e. it is a reduction-closed non-trivial equality which respects insensitivity. Assume given $\Gamma \vdash_{\mathsf{0}} P_i \triangleright A$ where, w.l.o.g. all types in Γ are paired. Suppose we have $P_1 \cong P_2$ and $P_1 \Downarrow_x$ but $\neg P_2 \Downarrow_x$. For simplicity, assume x has type $()^{?_1}$ (this does not lose generality since we can always transform an affine output of arbitrary form to this form). We now show this implies an arbitrary pair of two terms with the same type are equated in $\cong$, which contradicts the non-trivialness of $\cong$. We use the following property of typed terms:

Proposition 9. (i) *If $x(\vec{y}).P$ is typable and $\Gamma \vdash_\phi P \triangleright A$ then we have $\mathsf{md}(A) \subset \{?_1, ?_\omega\}$. Similarly for branching prefix.*
(ii) *If P is typable, then either $P \equiv \Pi_i P_i$ or $P \equiv (\boldsymbol{\nu}\, \vec{x})(\Pi_i P_i)|S$ where each P_i is an input-prefixed process and S is an output-prefixed prime process.*

Based on the above observations we first show all typed processes of the same type whose action modes are outputs are equated in $\cong$. By (i) this tells us all input-prefixed processes of the same type are equated (by the congruence of $\cong$) and by (ii) this tells us all typed terms of the same type are equated, contradicting $\cong$ is not trivial.

So take an arbitary $\Gamma \vdash_{\mathsf{0}} S \triangleright B$ such that (1) B only contains outputs and (2) $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \emptyset$. Then $(\boldsymbol{\nu}\, x)(x.S|P_i)$ $(i=1,2)$ is also well-typed. Further write $\overline{A}$ fore a dual of A and let $C[\ \ ]$ be given by:

$$C[\ \ ] \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \mathsf{fn}(A))(\Omega^{\Gamma}_{\overline{A},\mathtt{I}}|[\ \cdot\ ])$$

where the agent $\Omega^{\Gamma}_{\overline{A},\mathtt{I}}$ is as given in Appendix G. Note $C[(\boldsymbol{\nu}\, x)(x.S|P_i)]$ $(i = 1,2)$ has type B. Note $C[\ ]$ hides all and only free names of $P_{1,2}$, so that $\mathsf{fn}(C[(\boldsymbol{\nu}\, x)(x.S|P_i)]) = \mathsf{fn}(S)$. In particular $C[(\boldsymbol{\nu}\, x)(x.S|P_2)]$ is insensitive, thus by soundness we conclude $C[(\boldsymbol{\nu}\, x)(x.S|P_2)] \cong \Omega_{C,\mathsf{0}}$. Since P_1 does have an output at x, we also have:

$$C[(\boldsymbol{\nu}\, x)(x.S|P_1)] \ \longrightarrow\ S|(\boldsymbol{\nu}\, \mathsf{fn}(A))(\Omega^{\Gamma}_{\overline{A},\mathtt{I}}|P_1') \ \cong\ S.$$

This because $\vdash_{Imode} (\boldsymbol{\nu}\,\mathsf{fn}(A))(\Omega^{\Gamma}_{A,\mathtt{I}}|P_1') \triangleright \emptyset$ is insensitive, hence in particular equated with $\mathbf{0}$: then we use $\cong\supset\equiv$. By reduction closure we have $S \cong \Omega_{C,\mathtt{O}}$, as requied.

E Typed Transition

This appendix formally introduces typed transitions and studies their basic properties. We also introduce *composite transitions* for sequential processes and establish their basic properties.

E.1 Typed Transitions

Typed transitions describe the observations a sequentially typed observer can make of sequentially typed processes. Typed transitions are a proper subset of untyped transitions (as given in Appendix C.3) which however do not restrict the original τ-transition. To define typed transitions we use the following predicates.

(i) $A \vdash l$, when either: (1) $l = \tau$, or (2) $\mathsf{sbj}(l) = x$ and $px \in A$ such that : (2-a) $p \neq \bot$; and (2-b) if l is output then $p \neq \mathbf{!}_\omega$.
(ii) $\phi \vdash l$, when if l is input then $\phi = \mathtt{I}$ else $\phi = \mathtt{O}$.
(iii) $A, \phi \vdash l$, when both $A \vdash l$ and $\phi \vdash l$.

In (i), if $p = \bot$ then both input and output are supplied for the affine channel x, so no action with the outside at that channel should be possible; similarly when $p = \mathbf{!}_\omega$. (ii) makes sense since if P and Q interact in $P|Q$, then, if P is in $\mathtt{O}$-mode, Q should be in $\mathtt{I}$-mode. In this case no output of Q (input in P) can take place. Given the predicate $A, \phi \vdash l$, typed transitions are defined as follows.

$$\frac{\Gamma \vdash_\psi P \triangleright A, \quad P \xrightarrow{l} P', \quad A, \phi \vdash l, \quad \mathsf{bn}(l) \cap \mathsf{fn}(\Gamma) = \emptyset \qquad \text{if } l = \tau \text{ then } \psi = \phi \text{ else } \psi = \overline{\phi}}{\Gamma \vdash_\phi P \xrightarrow{l} \Gamma \cdot \mathtt{new}(l^\Gamma) \vdash_\psi P'}$$

Here $\mathtt{new}(l^\Gamma)$ denotes the name-type pairs introduced by l under Γ (for example, $\mathtt{new}(\overline{x}(yz)^\Gamma) = y : \tau_1, z : \tau_2$ if $\Gamma \vdash x : (\tau_1\tau_2)^{?_1}$). As expected, $\overline{\phi}$ returns the IO-mode dual to ϕ. Before proving Proposition 3, we need to show:

Proposition 10. *If* $\Gamma \vdash_\phi P \xrightarrow{l} \Delta \vdash_\psi Q$ *then* $\Delta \vdash_\psi Q$.

PROOF: Suppose $\Gamma \vdash_\phi P \xrightarrow{l} \Delta \vdash_\psi Q$. Then by definition, $\Gamma \vdash_\phi P \triangleright A$, $A, \phi \vdash l$ and $P \xrightarrow{l} Q$. The case $l = \tau$ is proven by using the Subject Reduction Theorem. For the other cases, we proceed by induction on the inference of the untyped transition $P \xrightarrow{l} Q$. Suppose $P \xrightarrow{l} Q$ is inferred by (IN). Then we can write $P \equiv a(\vec{y} : \vec{\tau}).Q$ and $P \xrightarrow{a(\vec{y})} Q$. By $\Gamma \vdash_{\mathtt{I}} P \triangleright A$, we have: $\Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{O}} Q \triangleright C^{\text{-}x}$ with $C/\vec{y} = \mathbf{?}A'$ and $A = A' \otimes \mathbf{!}_1 x$. Note that $\mathrm{new}(l^\Gamma) = \vec{y} : \vec{\tau}$ and $\overline{\mathtt{I}} = \mathtt{O}$. Hence

$\Gamma \cdot \mathrm{new}(l^{\Gamma}) \vdash_{\mathtt{O}} Q$ is well-typed. The other base cases (OUT), (BAR), (SEL), (IN!) and (BRA!) are similar. We also note that there is no typed transition $\Gamma \vdash_\phi P \stackrel{l}{\longrightarrow} \Delta \vdash_\psi Q$ such that $P \stackrel{l}{\longrightarrow} Q$ is derivable by (OUT-ξ,ξ_i). Since if so, by $\phi = \mathtt{O}$ and definition of $\phi \vdash l$, l should be output. However by Proposition 2 (ii-2), if $P \equiv \overline{a}(\vec{y})R$ and $R \stackrel{l}{\longrightarrow}$ with $\mathsf{fn}(l) \cap \{\vec{y}\} = \emptyset$, then l should be input. Other contextual rules such that parallel composition, restriction and structure rules are easy using the induction hypothesis. ■

Lemma 7. *Let* $\Gamma \vdash_\phi P$.

(i) (determinacy) *If* $\Gamma \vdash_\phi P \stackrel{l}{\longrightarrow} \Delta \vdash_\psi Q_i$ $(i = 1, 2)$ *then* $Q_1 \equiv_\alpha Q_2$.

(ii) (unique output) *If* $\Gamma \vdash_{\mathtt{O}} P \stackrel{l_i}{\longrightarrow}$ $(i = 1, 2)$ *then* $l_1 \equiv_\alpha l_2$.

(iii) (action and mode, 1) $\Gamma \vdash_{\mathtt{I}} P \stackrel{l}{\longrightarrow} \Delta \vdash_\phi Q$ *implies that* l *is input and* $\phi = \mathtt{O}$. *Conversely, if* $\Gamma \vdash_\phi P \stackrel{l}{\longrightarrow} \Delta \vdash_\psi Q$ *with* l *input then* $\phi = \mathtt{I}$.

(iv) (action and mode, 2) $\Gamma \vdash_{\mathtt{O}} P \stackrel{l}{\longrightarrow} \Delta \vdash_\phi Q$ *implies that either* $l = \tau$ *and* $\phi = \mathtt{O}$, *or* l *is output and* $\phi = \mathtt{I}$. *Conversely, if* $\Gamma \vdash_\phi P \stackrel{l}{\longrightarrow}$ *with* l *not being input then* $\phi = \mathtt{O}$.

(v) (IO-alternation) *Let* $\Gamma \vdash_\phi P \stackrel{l_1 l_2}{\Longrightarrow} \Delta \vdash_\psi Q$. *Then* (1) $\phi = \psi$, *and* (2) l_1 *is input iff* l_2 *is output and vice versa.*

PROOF: (i)...(iv) are easy by induction on typing rules. (v) is a direct corollary of (iii) and (iv). ■

E.2 Composite Transitions

Composite transitions record interactions of two sequential processes composed in parallel, including their mutual interactions and their interactions with the outside.

Definition 5. The relation $\Gamma \vdash_\phi P|Q \stackrel{s\langle t,u\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\,\vec{y})(P'|Q')$, called a *composite transition* of $P|Q$, is defined by the following induction. For the base case, we set $\Gamma \vdash_\phi P|Q \stackrel{\varepsilon\langle\varepsilon,\varepsilon\rangle}{\Longrightarrow} \Gamma \vdash_\phi P|Q$ if $\Gamma \vdash_\phi P|Q$. For the inductive step, assume $\Gamma \vdash_\phi P|Q \stackrel{s\langle t,u\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\,\vec{y})(P'|Q')$ and assume names in $\mathsf{bn}(l)$ to be fresh.

(Visible Action) if $P' \stackrel{l}{\longrightarrow} P''$ such that $l \neq \tau$, then $\Gamma \vdash_\psi P|Q \stackrel{sl\langle tl,u\rangle}{\Longrightarrow} \Gamma'' \vdash_{\psi''} (\boldsymbol{\nu}\,\vec{y})(P''|Q')$. Symmetrically when Q' has a non-τ-action.

(τ-action, 1) If $P' \stackrel{\tau}{\longrightarrow} P''$, then $\Gamma \vdash_\psi P|Q \stackrel{s\langle t,u\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\,\vec{y})(P''|Q')$. Symmetrically when Q' has a τ-action.

(τ-action, 2) If $P'|Q' \stackrel{\tau}{\longrightarrow} (\boldsymbol{\nu}\,\vec{z})(P''|Q'')$ from $P' \stackrel{l}{\longrightarrow} P''$ and $Q' \stackrel{\bar{l}}{\longrightarrow} Q''$, then $\Gamma \vdash_\psi P|Q \stackrel{s\langle tl,u\bar{l}\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\,\vec{y}\vec{z})(P''|Q'')$.

Since reduction is deterministic, s uniquely determines t and u in $\Gamma \vdash_\phi P|Q \stackrel{s\langle t,u\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\,\vec{y})(P'|Q')$.

F Term Trees and Enriched Transition

To establish the basic results in Section 4, we need to analyse the structure of typed terms and how it affects their dynamics. To do so we need tools to precisely specify and manipulate syntactic structures. This appendix offers a formal treatment of term trees and occurrences and introduces *enriched transitions*, which record not only transitions but also their generating redexes (subterms). These notions are closely related to the *proved transitions* of [11]. The readers may jump to the next section for the main analysis of typed transitions, referring back to this section as necessary.

F.1 Term Trees

We first introduce standard ideas from term algebra and formally define the signature for the calculus. For simplicity and without loss of generality, in this section we assume all indices in branching and selections to be in $\mathbb{N}$ (the set of natural numbers) and the corresponding indexing sets to be initial segments of $\mathbb{N}$. We shall also consistently neglect type annotations in typed terms in this section.

Definition 6.

(i) *A* signature Σ *is a set of pairs* (f,n) *(*$n \in \mathbb{N}\cup\{\mathbb{N}\}$*) such that* $(f,n),(f,n') \in \Sigma$ *implies* $n = n'$*. Here* f *is a* symbol *and* n *is* f*'s* arity*. We shall often write* $\Sigma(f) = n$ *to denote* $(f,n) \in \Sigma$*. We sometimes omit* f*'s arity.*
(ii) *Given a signature* Σ*, a* Σ-tree *is a partial function* $T : \mathbb{N}^* \to \Sigma$ *such that (1) if* $T(\sigma.\gamma)$ *is defined so is* $T(\sigma)$*, and (2) if* $T(\sigma) = (f,n)$ *then* $T(\sigma.i)$ *is defined iff* $0 \leq i < n-1$*. We write* $\xi, \xi', \ldots$ *for* occurrences *of a given tree, i.e. elements of its domain. Two occurrences* ξ_1 *and* ξ_2 *are* incomparable *if neither is a prefix of the other. The* meet *of* ξ *and* ξ'*, denoted* $\xi \sqcap \xi'$*, is the longest common prefix of* ξ *and* ξ' *(which can be the empty string).*
(iii) *We define a signature* Σ_π *as follows, with* $x, y, \ldots$ *ranging over names,* $i, j, \ldots$ *over* $\mathbb{N}$ *and* $I, J, \ldots$ *over the set of non-empty subsets of* $\mathbb{N}$*. A sequence of names in parenthesis is assumed to be pairwise distinct.*

$$\Sigma_\pi \;=\; \begin{array}{l}\{(\mathbf{0},0),\ ((\boldsymbol{\nu}\, x),1),\ (|,2),\ (x(\vec{y}),1),\ (x[\&_{i\in I}(\vec{y}_i)],I),\\ \ (\overline{x}(\vec{y}),1),(\overline{x}\mathtt{in}_i(\vec{y}),1),(!x(\vec{y}),1),\ (!x[\&_{i\in I}(\vec{y}_i)],I)\}\end{array}$$

From now on we shall concentrate on the signature Σ_π*.* Σ_π*-trees are often simply called* trees *from now on.*

The treatment easily extends to (one-hole and multiple-hole) contexts, which we omit. Clearly there is a one-to-one correspondence between Σ_π-trees and processes in the sense of Section 3.

The following two operations on trees are used to be able to define transitions on Σ_π-trees and to insert new restrictions into a tree. This is necessary to capture the effect of scope extensions induced by τ-actions.

Definition 7. (substitution and insertion)

(i) *Let* $\xi \in \mathtt{dom}(T)$. *Then* $T[T'/\xi]$ *denotes the result of substituting the occurrence* ξ *of* T *by* T'. *Formally:*

$$T[T'/\xi](\gamma) = \begin{cases} T'(\delta) & \gamma = \xi.\delta \\ T(\gamma) & \textit{otherwise} \end{cases}$$

(ii) *Let* T *be a tree,* ξ *be an occurrence in* T *and* $f_1, \ldots, f_n$ *be unary symbols* $(n \geq 0)$. *Then we write* $T\langle f_1...f_n/\xi\rangle$ *for the result of inserting the sequence of symbols* $f_1...f_n$ *at position* ξ. *Formally:*

$$T\langle f_1...f_n/\xi\rangle(\gamma) = \begin{cases} T(\xi.\delta) & \gamma = \xi.0^n.\delta \\ f_i & \gamma = \xi.0^i, 0 < i < n \\ T(\gamma) & \textit{otherwise} \end{cases}$$

F.2 Redexes

Hereafter we identify processes and Σ_π-trees, writing $P, Q, \ldots$ to denote them. We are mainly interested in subtrees (subterms) corresponding to occurrences of a given term, which we also write $P(\xi), Q(\xi'), \ldots$ if no confusion arises. Note that $P(\xi_1)$ is a subterm of $P(\xi_2)$ when ξ_2 is a prefix of ξ_1.

Definition 8. (continuation) *Given a prefixed* P, *an* action l of P *and the* l-continuation of P *are defined as follows.*

- *If* $P = x(\vec{y}).P'$, $x(\vec{y})$ *is its action, and* P' *is the* $x(\vec{y})$*-continuation. Similarly if* $P(\xi) = x[\&_{i\in I}(\vec{y}_i).P'_i]$, *each* $x\mathtt{in}_i(\vec{y}_i)$ *is its action and* P'_i *is the* $x\mathtt{in}_i(\vec{y}_i)$*-continuation.*
- *If* $P = \overline{x}(\vec{y})P'$, $\overline{x}(\vec{y})$ *is its action, and* P' *is the* $\overline{x}(\vec{y})$*-continuation. Similarly if* $P(\xi) = \overline{x}\mathtt{in}_i(\vec{y})P$, $\overline{x}\mathtt{in}_i(\vec{y})$ *is its action and* P' *is the* $\overline{x}\mathtt{in}_i(\vec{y})$*-continuation.*
- *If* $P = !x(\vec{y}).P'$, $x(\vec{y})$ *is its action, and* $(P'|!x(\vec{y}).P')$ *is the* $x(\vec{y})$*-continuation. Similarly if* $P(\xi) = !x[\&_{i\in I}(\vec{y}_i).P'_i]$, *each* $x\mathtt{in}_i(\vec{y}_i)$ *is its action and* $P'_i|!x[\&_{i\in I}(\vec{y}_i).P'_i]$ *is the* $x\mathtt{in}_i(\vec{y}_i)$*-continuation.*

We write $\mathsf{cont}(P, l)$ *for the* l*-continuation of* P *when* l *is an action of* P.

We also formalise the idea of *binding*. Given occurrences ξ and $\xi.i$ of P, the *binder of* ξ *over* $\xi.i$, denoted $\nu_P(\xi, i)$, is defined as follows:

$$\nu_P(\xi, i) = \begin{cases} \{\vec{y}\} & P(\xi) \text{ is of form } x(\vec{y}).P',\ \overline{x}(\vec{y})P',\ \overline{x}\mathtt{in}_i(\vec{y})P' \text{ or } !x(\vec{y}).P' \\ \{\vec{y}_i\} & P(\xi) \text{ is of form } x[\&_i(\vec{y}_i)P'_i] \text{ or } !x[\&_i(\vec{y}_i)P'_i] \\ \{y\} & P(\xi) \text{ is of form } (\boldsymbol{\nu}\, y)P' \\ \emptyset & \text{otherwise} \end{cases}$$

Note that, in the first case, we always have $i = 0$. For an input to interact with an appropriate output, each must "travel" up the Σ_π-tree towards the first node that is an ancestor of both the input and the output occurrence. The joint subject of the two actions must not be restricted or bound along the path from the input (or output) up to the said ancestor. The following definition aids in formalising that constraint.

Definition 9. *Let ξ_{base} be a prefix of ξ_{end}. We then define $\mathsf{res}_P(\xi_{base}, \xi_{end})$ by the following induction.*

$$\mathsf{res}_P(\xi_{base}, \xi_{end}) = \begin{cases} \emptyset & (\xi_{end} = \xi_{base}) \\ \nu_P(\xi_{base}\cdot\xi_{rest}, i) \cup \mathsf{res}_P(\xi_{base}, \xi_{base}\cdot\xi_{end}) & (\xi_{end} = \xi_{base}\cdot\xi_{rest}\cdot i) \end{cases}$$

We write $\mathsf{res}_P(\xi_{end})$ for $\mathsf{res}_P(\varepsilon, \xi_{end})$.

We are now ready to define redexes.

Definition 10. (i) *An* input-redex *(resp.* output-redex*) in P is an occurrence ξ of P such that $P(\xi)$ is input-prefixed (resp. output-prefixed). A redex in P is* active *if it is not a subterm of an input prefixed-term.*

(ii) *Two active redexes ξ_1 and ξ_2 in P are* complementary *if (1) $P(\xi_1)$ and $P(\xi_2)$ have actions l and $\bar{l}$ (where $\bar{l}$ is the dual of l); and (2) for l in (1) we have $\mathsf{sbj}(l) \notin \mathsf{res}_P(\xi_1 \sqcap \xi_2, \xi_i)$ for $i = 1, 2$.*

Above (3) formalises what has been alluded to above, that binders of interacting redexes can only be common ancestors of both participating redexes.

Lemma 8. *Let $\Gamma \vdash_\phi P \triangleright A$ below.*

(i) *If $\phi = \mathtt{O}$ then P has exactly one active output redex with subject x such that $\perp x \in A$ or $?_1 x \in A$ or $?_\omega x \in A$ or $!_\omega x \in A$.*

(ii) *If $\phi = \mathtt{I}$ then P has no active output redexes.*

(iii) *If $\{\xi_i\}$ $(i \in I)$ is the set of input redexes of P, then $i \neq j$ implies $\mathsf{sbj}(\xi_i) \neq \mathsf{sbj}(\xi_j)$. Further, for all $i \in I$, we have $\perp x_i \in A$ or $!_1 x_i \in A$ or $!_\omega x_i \in A$.*

(iv) *If $\phi = \mathtt{O}$, ξ_o is the unique output redex in P and for some $i \in I$: $\mathsf{sbj}(\xi_o) = \mathsf{sbj}(\xi_i) = x$ then $\perp x \in A$ or $!_\omega x \in A$.*

(v) *If P has an active output redex with subject x then $?_1 x \in A$ or $?_\omega x \in A$ or $\perp x \in A$ or $!_\omega x \in A$. The reverse implication does not hold.*

(vi) *$!_1 x \in A$ if and only if P has an active input redex with subject x which is not replicated.*

(vii) *$!_\omega x \in A$ if and only if P has an active replicated redex with subject x.*

(viii) *If P has two distinct active redexes ξ, ξ' such that $\mathsf{sbj}(\xi) = x = \mathsf{sbj}(\xi')$ then ξ and ξ' are complementary and either $\perp x \in A$ or $!_\omega x \in A$.*

(ix) *If $\perp x \in A$ and there is an active redex ξ in P with $\mathsf{sbj}(\xi) = x$ then there exists a complementary active redex for ξ in P.*

PROOF: All are easy inductions on the derivation of typing judgement. ∎

F.3 Enriched Transitions

Below we introduce enriched transitions of two kinds: $P \xrightarrow{l,\xi} Q$ indicating that $P \xrightarrow{l} Q$ where ξ is the occurrence firing the action and $P \xrightarrow{\xi_1,\xi_2} Q$ which indicates $P \xrightarrow{\tau} Q$ where ξ_1 and ξ_2 are the corresponding redexes giving rise to the τ-action. In (ii) below we write $P\langle(\boldsymbol{\nu}\, x_1..x_n)/\xi\rangle$ for $P\langle(\boldsymbol{\nu}\, x_1)(\boldsymbol{\nu}\, x_2)..(\boldsymbol{\nu}\, x_n)/\xi\rangle$ (cf. Definition 7 (ii)).

Definition 11. (enriched transitions)

(i) $P \xrightarrow{l,\xi} Q$ *with* $l \neq \tau$ *if, for* $P' \equiv_\alpha P$, *we have* $Q \equiv_\alpha P'[R/\xi]$ *where (1)* $R = \mathsf{cont}(P'(\xi), l)$ *and (2)* ξ *is an active redex such that* $\mathsf{sbj}(l) \notin \mathsf{res}_P(\xi)$.

(ii) $P \xrightarrow{\xi_1,\xi_2} Q$ *if, for* $P' \equiv_\alpha P$, *we have* $Q \equiv_\alpha P'[R_1/\xi_1][R_2/\xi_2]\langle(\boldsymbol{\nu}\,\vec{y})/\xi_1 \sqcap \xi_2\rangle$ *where (1)* $R_i = \mathsf{cont}(P'(\xi_i), l_i)$ $(i = 1,2)$ *with* $l_1 = \overline{l_2}$; *and (2)* ξ_1, ξ_2 *are two active and complementary redexes in* P.

Clearly $P \xrightarrow{l} Q$ with $l \neq \tau$ iff $P \xrightarrow{l,\xi} Q$ for some active redex ξ with action l in P. Similarly $P \xrightarrow{\tau} Q$ iff there are complementary active redexes ξ_1, ξ_2 in P such that their shared subject is not restricted in $\mathsf{res}_P(\xi_1 \sqcap \xi_2, \xi_i)$ for $i = 1, 2$. Also note that l is completely determined by P, ξ and Q in $P \xrightarrow{l,\xi} Q$ so we sometimes omit it.

G Visibility, Well-Bracketing and Innocence

In this appendix we establish basic facts about the behaviour of typed processes as stated in Section 4, namely visibility, well-bracketing and innocence. Although visibility can be established without using the Permutation Lemma (Lemma 1), for economy of presentation, we shall first prove the Permutation Lemma and use it to establish visibility.

G.1 Enabling Relations

We introduce two enabling relations on transition sequences, $\curvearrowright_{\mathsf{p}}$ and $\curvearrowright_{\mathsf{b}}$, which we shall use extensively in the following.

Definition 12. (enabling relations, 1)

(i) *We define the* prefixing relation $\curvearrowright_{\mathsf{p}}$ *as follows:* $P \xrightarrow{l_1,\xi_1} Q \curvearrowright_{\mathsf{p}} R \xrightarrow{l_2,\xi_2} S$ *when* $Q = R$, ξ_1 *is a prefix of* ξ_2 *and* l_1 *is an input action.*

(ii) *We define the* binding relation $\curvearrowright_{\mathsf{b}}$ *as follows:* $P \xrightarrow{l_1,\xi_1} Q \curvearrowright_{\mathsf{b}} R \xrightarrow{l_2,\xi_2} S$ *when* $Q = R$, ξ_1 *is a prefix of* ξ_2 *and* $\mathsf{sbj}(l_2) \in \mathsf{bn}(l_1)$.

We shall often write $l_1 \curvearrowright_{\mathsf{b}} l_2$ or $\xi_1 \curvearrowright_{\mathsf{b}} \xi_2$ for $P \xrightarrow{l_1,\xi_1} Q \curvearrowright_{\mathsf{b}} Q \xrightarrow{l_2,\xi_2} R$ when the processes in question are irrelevant or easily determined from the context. Similar conventions are used for the other relations on transitions.

Next we extend these relations to τ-transitions. Below we say σ is a δ*-shift prefix* of γ if $\sigma = \sigma'.\sigma''$ and $\gamma = \sigma'.\delta.\sigma''.\gamma'$ for some $\sigma', \sigma'', \gamma'$.

Definition 13. (enabling relations, 2)

(i) $P \xrightarrow{\xi_1,\xi_2} Q \curvearrowright_{\mathsf{p}} R \xrightarrow{l,\xi} S$ *when* $Q = R$, l *is an output and the input redex in* $\{\xi_1, \xi_2\}$ *is an* $\underbrace{0...0}_{n}$*-shift prefix of* ξ *with* n *the arity of the output redex in* $\{\xi_1, \xi_2\}$.

(ii) $P \xrightarrow{l,\xi} Q \curvearrowright_{\mathsf{p}} R \xrightarrow{\xi_1,\xi_2} S$ *when* $Q = R$, l *is an input action and* ξ *prefixes the output action in* $\{\xi_1, \xi_2\}$.

(iii) $P \xrightarrow{\xi_1,\xi_2} Q \curvearrowright_{\mathsf{p}} R \xrightarrow{\zeta_1,\zeta_2} S$ *when* $Q = R$ *and, assuming that* ξ_i *is an input redex and* ζ_j *and output redex, then* ξ_i *is an* $\underbrace{0...0}_{n}$*-shift prefix of* ζ_j *where, as above,* n *is the arity of the output redex in* $\xi_{1,2}$.

The relation $\curvearrowright_{\mathsf{b}}$ *is extended to* τ*-transition just as* $\curvearrowright_{\mathsf{p}}$*, but replacing prefixing with binding. Finally the* enabling relation $\curvearrowright$ *is the union of prefixing and binding:* $\curvearrowright = \curvearrowright_{\mathsf{b}} \cup \curvearrowright_{\mathsf{p}}$.

From now on we extensively (and often implicitly) use the following properties of enabling.

Lemma 9.

(i) *If* $\Gamma \vdash_\phi P \xrightarrow{l_1 l_2}$ *where* l_2 *is either output or* τ*-action, then* $l_1 \curvearrowright_{\mathsf{p}} l_2$.

(ii) *If* $\Gamma \vdash_\phi x(\vec{y}).P \xrightarrow{x(\vec{y})} \xrightarrow{\tau^*} \xrightarrow{l}$ *then* $x(\vec{y}) \curvearrowright^*_{\mathsf{p}} l$.

Proof: (i) is mechanical by induction. (ii) is a simple corollary of (i). ■

G.2 Permutation Lemma

When permuting actions we arrive at states which are not sequential. To deal with these states, we use processes which are typable by a typing system which is the sequential one except that it forgets IO-modes.

Definition 14. (affine processes and transition) *We write* $\Gamma \vdash P \triangleright A$*, or simply* $\Gamma \vdash P$*, for a typed process derived by typing rules of Figure 1 (together with Appendix A) which ignore all conditions on IO-modes. Accordingly we write* $\Gamma \vdash P \xrightarrow{l} \Delta \vdash Q$ *for the typed transition given as in Appendix E.1 except for ignoring the condition on IO-modes.*

Note that the newly introduced typed processes and transitions strictly include the original processes and transitions. Note also that we use the identical set of bases for typing. By the same reasoning as for Proposition 10, we observe:

Fact 2 $\Gamma \vdash P \xrightarrow{l} \Delta \vdash Q$ *implies* $\Delta \vdash Q$.

Note that if we write e.g. $\Gamma \vdash_\phi P \xrightarrow{l} \Delta \vdash_\psi Q$ then this indicates that *both* mentioned processes and transitions are sequential, while $\Gamma \vdash P \xrightarrow{l} \Delta \vdash Q$ indicates that we ignore IO-modes in both typability and transitions.

We need two properties from the newly introduced transitions. The second one pertains to "statelessness" of replication.

Lemma 10. *Let $\Gamma \vdash \xrightarrow{l_1} \ldots \xrightarrow{l_n} \Delta \vdash Q$. If $l_i \curvearrowright_{\mathsf{b}} l_j$ then one of the following must be the case: $l_i = \tau = l_j$ or $l_i, l_j \neq \tau$.*

Proof: No visible action can bind a redex in a τ-action, for otherwise the visible action would have to bind both redexes which would mean that both input and output types would occur bound by an input. This would violate sequentiality constraints on channel types. Similarly, a redex in a τ-action can never bind a visible action because the input and the dual output generating the τ-action both bind the same names which are restricted and remain so after launching the binding τ-action. Thus the bound visible action could not actually become visible. ■

Lemma 11. (one-step permutability) *Let $\Gamma \vdash P \xrightarrow{l_1}\xrightarrow{l_2} \Delta \vdash Q$ such that neither $l_1 \curvearrowright_{\mathsf{b}} l_2$ nor $l_1 \curvearrowright_{\mathsf{p}} l_2$. Then $\Gamma \vdash P \xrightarrow{l_2}\xrightarrow{l_1} \Delta \vdash Q$.*

Proof: There are nine cases to consider, with each l_i being either input, output or a τ-action. We abbreviate these cases to OO, IO, OI, II, $O\tau$, τO, $I\tau$, τI and $\tau\tau$. The first seven cases are immediately permutable since the redex(es) of l_2 are already active in P and are disjoint (not contained in) a redex of l_1. Among then we show two cases. The case II is permutable because if the l_i in $\Gamma \vdash P \xrightarrow{l_1}\xrightarrow{l_2} \Delta \vdash Q$ are both input then the redex for l_1 does not contain that of l_2 because (1) input cannot bind input (by the assumptions) and (2) no free input subject can be prefixed (by the typing rule for input). On the other hand the case $O\tau$ is permutable since output is asynchronous and an output redex cannot bind a redex of τ by Lemma 10. For the final three cases, from the assumptions the redex of l_2 cannot be properly contained in a redex of l_1. Therefore the only possibility that they are not disjoint is when they share the same redex, that is when both actions involve an identical replicated input. By rule induction of the derivation of transition this case is easily permutable because replication is persistent. ■

The exact unit of permutation in sequential transitions becomes clear from the following observation.

Lemma 12. *Whenever we have $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_1 l_2}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} Q$, this transition is induced by transitions of form $P \xrightarrow{l_1}\xrightarrow{\tau^*}\xrightarrow{l_2} Q$.*

Proof: We use Lemma 7 to conclude that l_1 must be an input action and hence the mode turns to the output mode. By Lemma 7, a process in output mode can only do τ or output, but if the latter takes place the mode turns to the input mode again, so only the last action can be an input. ■

A key lemma for permutability of sequential transitions follows.

Lemma 13. *Let* $\Gamma \vdash_{\mathtt{O}} P \stackrel{l_1^{\mathtt{O}} l_2^{\mathtt{I}} \tau^* l_3^{\mathtt{O}}}{\longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ *such that* $l_1 \not\curvearrowright_{\mathtt{b}} l_2$. *Then for some* P_1, P_1', P_2, P_2' *and* $\vec{x}$ *such that* $P \equiv (\nu \vec{x})(P_1|P_2)$ *and* $Q \equiv (\nu \vec{x})(P_1'|P_2')$, *we have* $P_1 \stackrel{l_1^{\mathtt{O}}}{\longrightarrow} (\nu \vec{x})P_1'$ *and* $P_2 \stackrel{l_2^{\mathtt{I}} \tau^* l_3^{\mathtt{O}}}{\longrightarrow} (\nu \vec{x})P_2'$.

PROOF: By Fact 1, we can set $P \equiv (\boldsymbol{\nu}\, \vec{z})(P_1|P_2)$ such that P_1 is an output-prefixed process (whose subject is not among $\vec{z}$ and) which has no free input subjects, either active or inactive. Let $\Gamma \vdash_{\mathtt{O}} P \stackrel{l_1^{\mathtt{O}}}{\longrightarrow} \Delta \vdash_{\mathtt{I}} (\boldsymbol{\nu}\, \vec{z})(P_1'|P_2)$ from $P_1 \stackrel{l_1^{\mathtt{O}}}{\longrightarrow} P_1'$. Then all active subjects (which are all input by Lemma 7) in P_1' are bound by l_1. This and $l_1 \not\curvearrowright_{\mathtt{b}} l_2$ imply that the redex for $P_1'|P_2 \stackrel{l_2^{\mathtt{I}}}{\longrightarrow}$ should come from P_2. By Lemma 10 the redexes for the subsequent τ-actions cannot be in P_1'.■

We also observe:

Lemma 14. *Let* $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_1^{\mathtt{I}} \tau^n l_m^{\mathtt{O}} l_{m+1}^{\mathtt{I}} \tau^{n'} l_{m'}^{\mathtt{O}}}{\longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ *such that* $l_m \not\curvearrowright_{\mathtt{b}} l_{m+1}$. *Then whenever* l_i *is a* τ*-action before* l_m *and* l_j *is a* τ*-action after* l_{m+1} *we have* $l_i \not\curvearrowright_{\mathtt{b}} l_j$.

PROOF: Assume l_j is the first τ-action after l_{m+1} which is bound by some τ-action before l_m. We show by partial induction on $k \in \{m+2, \ldots, j-1\}$ that there is $k' \in \{m+2, \ldots, k\}$ and some $k'' \in \{m+1, \ldots, k'\}$ such that $l_{k''} \curvearrowright_{\mathtt{b}} l_{k'}$. This contradicts Lemma 10 when $k = m+2$. We show the reasoning for $k = j$, which is identical with the induction step. By Lemma i, the output redex of l_j is contained in the input redex, say ξ, of l_{j-1}. In the corresponding occurrence at the time of the action l_i, ξ's subject should be bound for otherwise the input redex of l_i prefixes a free input subject, which is not possible by the typing rules. Therefore $l_{j-1} = l_{m+1}$ cannot be the case since the subject of l_{m+1} should already be free in P by $l_m \not\curvearrowright_{\mathtt{b}} l_{m+1}$. By our hypothesis, the subject of the redex of l_{j-1} is bound either (1) by a(n output) redex of a τ-action after l_{m+1}, which is contained by some input redex of, say $l_{k''}$, or (2) by a restriction inside the input redex of l_{j-2}, which we set $l_{k''}$. Note that the same condition as l_j applies to $l_{k''}$ in both cases so that we can use induction. ■

We can now prove the Permutation Lemma 1, which we state again for reference.

Lemma 15. (permutation) *Let* $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_1 l_2 l_3 l_4}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ *such that* $l_1 \not\curvearrowright_{\mathtt{b}} l_4$ *and* $l_2 \not\curvearrowright_{\mathtt{b}} l_3$. *Then* $\Gamma \vdash_{\mathtt{I}} P \stackrel{l_3 l_4 l_1 l_2}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} Q$.

PROOF: By Lemma 12 $\Gamma \vdash P \stackrel{l_1 \tau^* l_2 l_3 \tau^* l_4}{\longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ can be permuted into $\Gamma \vdash P \stackrel{l_3 \tau^* l_4 l_1 \tau^* l_2}{\longrightarrow} \Delta \vdash_{\mathtt{I}} Q$ (the latter is a sequential transition sequence by easy induction). By our assumption and by Lemmas 13 and 14, no action including and between l_3 and l_4 is either bound or prefixed by action including and between l_1 and l_2. We can now use Lemma 11 to permute each of the actions including and between l_3 and l_4 to the position immediately preceding l_1, starting from l_3, resulting in the required transition sequence. ■

G.3 Components and Well-Knit Sequences

Here we introduce the notion of components in a visible transition sequence (i.e. its subsequence whose actions are related by $\curvearrowright_{\mathsf{p}}^*$ and $\curvearrowright_{\mathsf{b}}^*$) and show that any sequential transition can be rearranged into a concatenation of sequences each being a component (by using the Permutation Lemma).

Definition 15. *Given a sequence $l_1...l_n$, its (set of)* component boundaries, *$\mathsf{cb}(l_1...l_n)$ is defined as follows.*

$$\mathsf{cb}(l_1...l_n) = \begin{cases} \{i \mid \mathsf{md}(l_i) = \mathtt{I}, \forall j.l_j \not\curvearrowright_{\mathsf{b}} l_i\} & \mathsf{md}(l_1) = \mathtt{I} \\ \{i \mid \mathsf{md}(l_i) = \mathtt{I}, \forall j.l_j \not\curvearrowright_{\mathsf{b}} l_i\} \cup \{1\} & \mathsf{md}(l_1) = \mathtt{O} \end{cases}.$$

Fix a sequence $\sigma = l_1...l_n$. The component *of l_i in σ, $\mathsf{comp}_\sigma(l_i)$, is defined next.*

$$\mathsf{comp}_\sigma(l_i) = \begin{cases} i & i \in \mathsf{cb}(l_1...l_n), \\ \mathsf{comp}_\sigma(l_j) & i \notin \mathsf{cb}(l_1...l_n), l_j \curvearrowright_{\mathsf{b}} l_i, \\ \mathsf{comp}_\sigma(l_{i-1}) & i \notin \mathsf{cb}(l_1...l_n), l_j \not\curvearrowright_{\mathsf{b}} l_i. \end{cases}$$

We shall often abbreviate $\mathsf{comp}_\sigma(l)$ to $\mathsf{comp}(l)$.

Definition 16. *Let $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$. A (contiguous) substring γ of σ is a* contiguous component *of σ if (1) γ is closed under binding, that is, $\gamma_j \curvearrowright_{\mathsf{b}} \sigma_j$ implies $\sigma_j = \gamma_k$ for some k; (2) if an input l is in γ and it is not the last action of σ then the next output is also in γ; and (3) it is a maximal such subsequence, i.e. no proper superstring of γ in σ has the properties (1) and (2).*

Definition 17. *Let $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$. We say σ is* blockwise well-knit *if $\sigma = \gamma_1...\gamma_n$ where*

- *each γ_i is a contiguous component of σ,*
- *if a contiguous component has a free (with respect to σ) input then it is the first element of the contiguous component.*

Lemma 16. *Let $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$. Then there is a componentwise permutation σ' of σ which is blockwise well-knit.*

Proof: By repeated applications of Lemma 1. ■

G.4 Visibility

We first reiterate the definitions of both input views and output views.

Definition 18. *The* input view *of $l_1...l_n$, $\ulcorner l_1...l_n \urcorner^{\mathtt{I}}$, is defined as follows.*

$$
\begin{array}{lll}
\ulcorner \epsilon \urcorner^{\mathtt{I}} & = \emptyset & \\
\ulcorner \sigma.l_n \urcorner^{\mathtt{I}} & = \{n\} \cup \ulcorner \sigma \urcorner^{\mathtt{I}} & \mathsf{md}(l_n) = \mathtt{I} \\
\ulcorner \sigma.l_n \urcorner^{\mathtt{I}} & = \{n\} & \mathsf{md}(l_n) = \mathtt{O}, \forall i.l_i \not\curvearrowright_{\mathsf{b}} l_n \\
\ulcorner \sigma_1.l_i.\sigma_2.l_n \urcorner^{\mathtt{I}} & = \{i, n\} \cup \ulcorner \sigma_1 \urcorner^{\mathtt{I}} & \mathsf{md}(l_n) = \mathtt{O}, l_i \curvearrowright_{\mathsf{b}} l_n
\end{array}
$$

The corresponding output view, *$\ulcorner l_1...l_n \urcorner^{\mathtt{O}}$ is defined dually.*

$$
\begin{array}{lll}
\ulcorner \epsilon \urcorner^{\mathtt{O}} & = \emptyset & \\
\ulcorner \sigma.l_n \urcorner^{\mathtt{O}} & = \{n\} \cup \ulcorner \sigma \urcorner^{\mathtt{O}} & \mathsf{md}(l_n) = \mathtt{O} \\
\ulcorner \sigma.l_n \urcorner^{\mathtt{O}} & = \{n\} & \mathsf{md}(l_n) = \mathtt{I}, \forall i.l_i \not\curvearrowright_{\mathsf{b}} l_n \\
\ulcorner \sigma_1.l_i.\sigma_2.l_n \urcorner^{\mathtt{O}} & = \{i, n\} \cup \ulcorner \sigma_1 \urcorner^{\mathtt{O}} & \mathsf{md}(l_n) = \mathtt{I}, l_i \curvearrowright_{\mathsf{b}} l_n
\end{array}
$$

Here $\sigma = l_1...l_n$ and $\sigma_1 = l_1...l_{i-1}$ are non-empty strings and $\sigma_2 = l_{i+1}...l_n$.

Convention 1 *Given $\ulcorner l_1...l_n \urcorner^{\mathtt{O}}$, there is the unique subsequence of $l_1...l_n$ induced by $\ulcorner l_1...l_n \urcorner^{\mathtt{O}}$. We shall identify the two. Dually for the input view.*

The *output visibility* and *input visibility* have been defined in the main section: the empty sequence ε is output visible; a non-empty sequence $\sigma = l_1...l_n$ is output visible if each proper prefix of σ is output visible and if $\mathsf{md}(l_n) = \mathtt{O}$ and $l_i \curvearrowright_{\mathsf{b}} l_n$ then $i \in \ulcorner l_1...l_n \urcorner^{\mathtt{O}}$. Dually for input visibility. Below we say a sequence σ *extends* a sequence σ' if σ' is a postfix of σ. We observe:

Lemma 17. (i) *If $\sigma_1.\sigma$ extends $\sigma_2.\sigma$, then σ_1 extends σ_2.*
(ii) *If $\sigma' = l'_1...l'_n$ and $\sigma = l_1...l_m l'_1...l'_n$ such that σ extends σ'. Then*
  (a) $l'_i \in \mathsf{cb}(\sigma)$ implies $l'_i \in \mathsf{cb}(\sigma')$. It is not *the case that $\mathsf{cb}(\sigma) \subseteq \mathsf{cb}(\sigma')$.*
  (b) $\mathsf{comp}_{\sigma'}(l'_i) \subseteq \mathsf{comp}_{\sigma}(l'_i)$.
(iii) *If σ extends σ' then $\ulcorner \sigma' \urcorner^{\mathtt{O}} \subseteq \ulcorner \sigma \urcorner^{\mathtt{O}}$. The dual result holds for the input view.*

Proof: The first two are straightforward from the definitions. For (iii) assume that $\sigma' = l'_1...l'_n$ and $\sigma = l_1...l_m l'_1...l'_n$. We proceed by induction on m. The base case $\sigma = \sigma'$ is immediate. For the inductive step consider $\sigma'' = l_2...l_m l'_1...l'_n$. By (IH) we know that $\ulcorner \sigma'' \urcorner^{\mathtt{O}} \subseteq \ulcorner \sigma \urcorner^{\mathtt{O}}$ and similarly for the output view. We conclude $\ulcorner \sigma' \urcorner^{\mathtt{O}} \subseteq \ulcorner \sigma'' \urcorner^{\mathtt{O}}$ from the following fact: $\ulcorner \gamma \urcorner^{\mathtt{O}} \subseteq \ulcorner l.\gamma \urcorner^{\mathtt{O}}$ whether there is a binding $l \curvearrowright_{\mathsf{b}} l_i$ into γ or not. That fact is proved by straightforward induction on the length of γ. ■

Lemma 18. *Given a sequence $\sigma = l_1...l_n$,*

- *σ has only one component if and only if it has no free inputs or exactly on, at l_1.*
- *Assume that σ has just one component and $n > 0$ then $1 \in \ulcorner \sigma \urcorner^{\mathtt{O}}$.*

PROOF: The first is immediate from the definitions and the second is proved by induction on the length of σ. ■

Definition 19. *Let* $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$. *A* componentwise permutation *of* σ *is a string* σ' *of visible actions such that* $\Gamma \vdash_\phi P \stackrel{\sigma'}{\Longrightarrow} \Delta \vdash_\psi Q$ *can be obtained from* $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$ *by zero or more applications of the Permutation Lemma 1.*

Lemma 19. *Let* $\Gamma \vdash_\phi P \stackrel{\sigma l}{\Longrightarrow} \Delta \vdash_\psi Q$ *and assume that* $\sigma_1' l \sigma_2'$ *is a componentwise permutation of* σ. *Then* $\ulcorner \sigma l \urcorner^{\mathsf{O}} = \ulcorner \sigma_1' l \urcorner^{\mathsf{O}}$.

PROOF: Lemma 1 allows permutation of pairs of adjacent actions in visible typable traces only if they are in the different components. On the other hand, $\ulcorner \urcorner^{\mathsf{O}}$ chases binding pointers $\curvearrowright_{\mathsf{b}}$, so $\ulcorner \urcorner^{\mathsf{O}}$ will always be within one component and is unaffected by the permutations Lemma 1 permits. ■

Immediately:

Corollary 3. *Let* $\Gamma \vdash_\phi P \stackrel{\sigma}{\Longrightarrow} \Delta \vdash_\psi Q$ *and assume that* σ' *is a componentwise permutation of* σ. *Then* σ *is output visible if and only if* σ' *is output visible.*

Thus considering (output) visibility in general sequential transitions is the same thing as considering it in well-knit sequences. We now observe:

Proposition 11. *If* $\Gamma \vdash_\phi P|Q \stackrel{s\langle t,u\rangle}{\Longrightarrow} \Gamma' \vdash_{\psi'} (\boldsymbol{\nu}\, \vec{y})(P'|Q')$ *and* s *is well-knit and output (input) visible. Then* t *and* u *are output (input) visible.*

PROOF: We treat the case when $\phi = \mathtt{I}$. The case when $\phi = \mathtt{O}$ is similar. At each step, an output is immediately output-visible for both the whole term and its components if the previous sequence is input-visible. Hence we have only to check that (1) each input of a residual of (say) P is input-visible if it is so for the whole term, and that (2) the interaction between two components results in output visible sequences. The proof uses the so-called *switching condition.* As noted in the main section, we first introduce the board presentation of the above composite transition. It consists of four rows and n columns. In the first row, we write P's visible transitions; the second row is P's interaction with Q; the third row is Q's interaction with P (which is dual to the second row): finally we write Q's visible transition. Note that actions in each column occur in one of the three forms in the board presentation:

(1) In the first row only.
(2) In the second and third rows, dual to each other.
(3) In the fourth row only.

We now prove this, together with one of the most basic properties of functional interaction propounded in games semantics, the switching condition. Below by a *component process* we mean one of a pair of processes (i.e. P or Q in the above case).

Claim. (switching condition) In the board presentation of composite transition, whenever an action of a component process moves from one row to another, then the moving action is an output.

PROOF: The proof is by induction, showing at the same time:

(a) If an action is done in the first (resp. the fourth) row then its next action cannot be done in the fourth (resp. the first) row;
(b) Each time an action is done in a single row, the mode of the other component always stays in I, so that the mode of the whole term is determined by that of the former process (it is then immediate that the whole term has mode O where input and output alternate in each component when actions are at the second and third rows); and
(c) When the input view starting from any output in one of the first/fourth rows only involves actions in that row.

Let P and Q be component processes whose actions are recorded in the first-second rows and third-fourth rows, respectively. W.o.l.g we assume that both start in I-mode (other cases are covered by starting from the intermediate point below). Below residuals of P and Q are *also* designated by P and Q.

Base case. Assume P's action is first recorded. Then Q is clearly in I-mode.
Within the first row. Assume there are two consecutive actions by P in the first row. Let it be input. Then the next is output. Since Q stays as before, its mode does not change, hence (b) is trivial. For (c) if the output is free it trivially holds. If not it holds by induction by the switching condition (note when we move back to the binding input, then the immediately preceding output should stay in the same row).
Within the second/third rows. In this case (a)(b)(c) are vacuous.
Change from the first to another. Assume there are two consecutive actions moving from the first row to another. We show (a) together with the switching condition. Assume first the actions are output-input. Then the input-view starting from it stays in the first row. Since the whole sequence is well-knit no free input is possible. This violates the visibility, hence this case is not possible. This shows the switching condition. Since the mode of Q is I-mode the output cannot be done at the fourth row, showing (a).
Change from the second to another. Assume there are two actions, the first one in the second row by P and the second one in the first or fourth row. Assume the first one is output. Immediately the mode of Q as well as the whole term is O. The only possible visible action is thus an output by Q at the fourth row, establishing the switching condition, and (b) immediately holds. If the second action by Q is free, (c) is vacuous. If not, it goes to some action in the fourth row, from where we can use induction by the switching condition, hence (c) holds.

Using exactly the same reasoning for the dual cases, this concludes the proof of the claim. The reasoning for output visibility of the resulting visible sequence precisely follows the known reasoning in game semantics (cf. [28], A.13 of [24], [32]). ■

The result should also hold when s is not well-knit: however the proof may become more complex. We are now ready to prove visibility of sequential transitions.

Theorem 8. *If $\Gamma \vdash_\sigma P \stackrel{s}{\Longrightarrow} \Delta \vdash_\psi Q$ and s is input visible then s is output visible.*

PROOF: Let $\Gamma \vdash_\phi P \stackrel{s}{\longrightarrow} \Delta \vdash_\psi Q$. By Corollary 3 we safely assume s is well-knit and show it is output visible if it is input-visible by induction on the derivation of $\Gamma \vdash_\phi P$. The base case, the weakening rules and restriction are trivial. The case when $\Gamma \vdash_\phi P \stackrel{\text{def}}{=} P_1|P_2$ is direct from Corollary 3 and Proposition 11 (note that we use input visibility). For output prefixes, by Fact 1 we can safely assume P to have no free active subject except that of the outer-most output prefix. Thus s is of form ls' where l is output where s' is the transition sequence of the continuation of P. By Lemma 17 (iii) s' is input visible hence, by induction hypothesis, s' is output visible and consequentially, so is s. For input prefixes, suppose P is input prefixed. Then we can write, by well-knitness, that $s = ls'$ where the subject of l never occurs again in s', so that it is a transition sequence of the continuation of P, hence by induction hypothesis it is output visible. If l does not bind any action in s' then the output view of each output in s' is contained in that in s hence s is output visible. If it does bind an output action in s', say l', then the redex of l' is contained in that of l. Since s is well-knit, no free input subject occurs in s except in the initial l. By easy induction on the length of such an intermediate sequence this implies the output view of l' in s reaches l. ■

G.5 Well-Bracketing

The following lemmas are known in game semantic literature, if not in the general setting we prove them here.

Lemma 20. *Let $\Gamma \vdash_\phi P \stackrel{sl}{\longrightarrow}$ be such that (1) l is output, (2) s is well-bracketing, and (3) the component of l in sl is well-bracketing. Then sl is well-bracketing.*

PROOF: If either l is of mode $?_\omega$ (i.e. is "(") or $s = \varepsilon$, then sl is trivially well-bracketing. Suppose l is an of mode $?_1$ (i.e. is "]") and $s \neq \varepsilon$. We first consider the case l is free. Let the immediately preceding input be l'. By the definition of component, l and l' are in the same component. Since this component is well-bracketing, l' is an answer of type ")". We now argue that the O-view of s has one of the following two forms, where, in both cases, the last ")" is l'.

Case (A): "()..()()", with the initial "(" being initial in s; or
Case (B): ")..()()", with the initial ")" free but not necessarily initial in s.

In other words, the view does not contain "[". Because all actions between each matching pair of "(" and ")" in s should match in bracketing (by s being well-bracketing), and because the prefix of s before the initial ")" in the case of B should not contain an unanswered question, this proves sl is well-bracketing. We proceed by induction on the length of s.

(s is of length 1). Then l' is free so that it has form (B).

(s is of length $n+1$ assuming the property holds for length less than n). Let $s = l_1 s'$. By induction hypothesis either (A) or (B) holds in s'. Suppose the stated property holds up to length n and let $s = l_1 s'$. By induction hypothesis either (A) or (B) holds in s'. If (A) holds in $\ulcorner s' \urcorner^{0}$ then l_1 can only be ")" since sl should be well-bracketing. Hence s has form (A), and we are done with this case. If on the other hand $\ulcorner s' \urcorner^{0}$ has form (B), then either l_1 binds ")" that is initial in $\ulcorner s' \urcorner^{0}$, in which case we have the case (A) as a whole (note "]", i.e. $?_1$, can never bind ")", i.e. $!_1$), or l_1 does not bind such ")", in which case we still have the form (B), hence as required.

This proves the case when l is free in s. If not, i.e. if $sl = s_1 l'' s_2 l$ such that $l'' \curvearrowright_{\mathsf{b}} l$. Then, precisely following the same reasoning, we can show the postfixes of $\ulcorner s_2 \urcorner^{0}$ have form (A) or (B), by induction of their length. Since the output visibility implies the output view of $l'' s_2$ should reach l'', $\ulcorner s_2 \urcorner^{0}$ as a whole should have form (A), so that $l'' s_2 l$ has completly matching pairs of brackets. Immediately sl is well-bracketing, and we are done. ■

Another key result follows.

Lemma 21. *Let $\Gamma \vdash_{\phi} P|Q \stackrel{sl\langle u,t\rangle}{\Longrightarrow} \Delta \vdash_{\mathtt{I}} (\boldsymbol{\nu}\, \vec{y})(P'|Q')$ such that (1) l is output, (2) u, t and s are well-bracketing. Then s is well-bracketing.*

PROOF: By the standard parity argument, cf. [28], A.13 of [24], [32]). ■

We are now ready to establish Proposition 5.

Proposition 12. $\Gamma \vdash_{\phi} P$ *is well-bracketing.*

PROOF: By induction on typing rules, using the equivalence between well-knit transitions and non-well-knit ones in terms of legality. The inaction is trivial since it has no transitions. Similarly for the weakening rules which do not change transitions. For hiding, by the definition of typed transitions, $\Gamma \vdash_{\phi} (\boldsymbol{\nu}\, x : \alpha)P$ has the same well-knit transitions as $\Gamma \cdot x : \alpha \vdash_{\phi} P$ except for those which start from inputs at x when $!_{\omega} \in \mathsf{md}(\alpha)$, hence as required. Parallel composition holds by Lemma 21. For the prefixing rules we only present the case of unary prefixes. Branching prefixes can be dealt with similarly. For output, we again assume that their abstraction covers all active free names of the prefix's continuation, cf. Fact 1. Thus, if $\Gamma \cdot x : \tau \vdash_{\mathsf{O}} \overline{x}(\vec{y} : \vec{\tau})P \stackrel{s}{\longrightarrow}$, then we can assume that $s = \overline{x}(\vec{y})s'$ such that $\Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P \stackrel{s'}{\longrightarrow}$ where s' is well-bracketing and that s itself is well-knit. If $\mathsf{md}(\tau) = ?_1$ then s is immediately well-bracketing. If $\mathsf{md}(\tau) = ?_{\omega}$ then, if $x(\vec{y})$ binds, or equivalently: is answered by an action in s then let $s' = s'_0 l' s'_1$ where

l' is that action. By well-bracketing of s', we know that s'_0 has no unanswered questions and that s'_1 itself is well-bracketing. By an argument similar to that in the proof of Lemma 20, we can show, by induction on the length of s'_1, that s'_1 is of form "[..][..][..]" where each pair of "[" and "]" is matching. Since s'_1 itself is well-bracketed, so are the actions in-between. Thus $x(\vec{y})s'_1l'$ (hence s) is well-bracketed, as required. The case when $x(\vec{y})$ does not bind any action in s is the same as the above setting $s' = s'_0$. Finally, for input prefixing rules, if the prefix is of mode $!_1$ (that is: of type ")") there is nothing to prove. If not, let $\Gamma \cdot x : \tau \vdash_{\mathsf{0}} !x(\vec{y}:\vec{\tau})P \xrightarrow{s}$ and assume s to be well-knit. Thus $s = x(\vec{y})s'$ where x is the only free input. Assume $s' = s'_0l's'_1$ where $x(\vec{y})$ is answered by l' and, by s' being well-bracketing, s'_1 is well-bracketing. By the typing rule there is no free action of mode $?_1$ (i.e. of type "]") in s' nor is there any free input action. Thus, in particular, no free answers occur in s'_0. Hence, by well-bracketing, these actions have matching parenthesis as required. Finally, the case when there is no such l' is the same because s'_0 has no free answers. ■

G.6 Innocence

Lemma 22. *Let* $\Gamma \vdash_\phi P \stackrel{t}{\Longrightarrow}$ *be well-knit and* $s = \ulcorner t \urcorner^{\mathsf{0}}$. *Then* $\Gamma \vdash_\phi P \stackrel{s}{\Longrightarrow}$.

PROOF: Assume w.l.o.g. $\phi = \mathrm{I}$. We show $\Gamma \vdash_\phi P \stackrel{tt'}{\Longrightarrow}$ with t of even length and $s = \ulcorner t \urcorner^{\mathsf{0}}$ implies $\Gamma \vdash_\phi P \stackrel{st'}{\Longrightarrow}$ by induction on the length of t. When $t = \varepsilon$ the result trivially holds. When $t = t_0 l_n^o$, we can write $t = t'_0 l^i_{n-1} l^o_n$. If l_{n-1} is free, by well-knitness $t'_0 = \varepsilon$ hence done. If not, we have $t'_0 = t''_0 l_i t''_1$ where $i \curvearrowright_{\mathsf{b}} n-1$, that is $\ulcorner t \urcorner^{\mathsf{0}} = s_0 l_{n-1} l_n$ where $s_0 = \ulcorner t''_0 l_i \urcorner^{\mathsf{0}}$. By (IH), we have $\Gamma \vdash_\phi P \stackrel{s_0 t''_1 l_{n-1} l_n}{\longrightarrow}$. If $t''_1 = \varepsilon$ we are done. If not, by visibility l_n is either free or is bound from an action in s_0, hence we can repeatedly use the Permutation Lemma to obtain $\Gamma \vdash_\phi P \stackrel{s_0 l_{n-1} l_n}{\longrightarrow}$, as required. ■

We can now prove Proposition 6.

Proposition 13. (innocence) *Let* $\Gamma \vdash_\psi P \stackrel{s_i l_i}{\Longrightarrow}$ $(i = 1, 2)$ *such that:* (1) *both sequences are legal;* (2) *both* l_1 *and* l_2 *are outputs; and* (3) $\ulcorner s_1 \urcorner^{\mathsf{0}} \equiv_\alpha \ulcorner s_2 \urcorner^{\mathsf{0}}$. *Then we have* $\ulcorner s_1 \urcorner^{\mathsf{0}} \cdot l_1 \equiv_\alpha \ulcorner s_2 \urcorner^{\mathsf{0}} \cdot l_2$.

PROOF: We can safely assume that both sequences are well-knit. Let $\Gamma \vdash_\psi P \stackrel{s_i l_i}{\Longrightarrow}$ $(i = 1, 2)$ with the above condition. W.l.o.g. let $t = \ulcorner s_1 \urcorner^{\mathsf{0}} = \ulcorner s_2 \urcorner^{\mathsf{0}}$. By Lemma 22 and noting $tl_i = \ulcorner s_i l_i \urcorner^{\mathsf{0}}$, we have $\Gamma \vdash_\psi P \stackrel{tl_i}{\Longrightarrow}$. By Lemma 3 (i) there is at most one output action possible from a given process, hence we know $l_1 \equiv_\alpha l_2$, as required. ■

H Context Lemma

This appendix introduces and studies a basic preorder on processes which can be characterised either as the may-preorder, as the simulation preorder or as the set inclusion of the innocent function representation. We also introduce the induced equivalence.

H.1 Preorder $\sqsubseteq_{\mathsf{seq}}$

Definition 20. We define a typed relation $\sqsubseteq_{\mathsf{seq}}$ as follows: $\Gamma \vdash_\phi P_1 \sqsubseteq_{\mathsf{seq}} P_2$ if $\Gamma \vdash_\phi P_1 \stackrel{s}{\Longrightarrow}$ implies $\Gamma \vdash_\phi P_2 \stackrel{s}{\Longrightarrow}$.

Immediately, $\sqsubseteq_{\mathsf{seq}}$ is a preorder. Further by the determinacy of $\longrightarrow$ we have $\longrightarrow \cup \longrightarrow^{-1} \subset \sqsubseteq_{\mathsf{seq}}$. Also we can restrict traces considered in the definition of $\sqsubseteq_{\mathsf{seq}}$ to well-knit ones (or even output views), by the characterisations we have given so far. Another observation is that $\sqsubseteq_{\mathsf{seq}}$ only pertains to output, because of the following:

Fact 3 *Let* $\mathbf{R}$ *be any typed relation and* $\Gamma \vdash_{\mathtt{I}} P\mathbf{R}Q$*. Then* $P \stackrel{l}{\longrightarrow}$ *implies* $Q \stackrel{l}{\longrightarrow}$*.*

This is because enabled input actions are determined by action types which are shared by processes related by a typed relation. We now prove:

Proposition 14. $\sqsubseteq_{\mathsf{seq}}$ *is a typed precongruence.*

Proof: We use rule induction on typing rules. The base case, $\Gamma \vdash_\phi \mathbf{0} \sqsubseteq_{\mathsf{seq}} \mathbf{0}$, is trivial. Similarly for the weakening rules. For the other rules:
(Res) Assume $\Gamma \cdot x : \alpha \vdash_\phi P_1 \sqsubseteq_{\mathsf{seq}} P_2$ such that $\Gamma \vdash (\boldsymbol{\nu}\, x : \alpha)P_i$ is well-typed. Assume $\Gamma \vdash (\boldsymbol{\nu}\, x : \alpha)P_1 \stackrel{s}{\Longrightarrow}$. W.l.o.g. we assume $x \notin \mathsf{fn}(s)$. Since all actions in s are never prohibited by hidden px, we know $\Gamma \cdot x : \alpha \vdash P_1 \stackrel{s}{\Longrightarrow}$. By induction hypothesis $\Gamma \cdot x : \alpha \vdash P_2 \stackrel{s}{\Longrightarrow}$, hence $\Gamma \vdash (\boldsymbol{\nu}\, x : \alpha)P_2 \stackrel{s}{\Longrightarrow}$ as required.
(Par) Assume $\Gamma \vdash_\psi P_1 \sqsubseteq_{\mathsf{seq}} P_2$ s.t. $\Gamma \vdash_\phi P_i|R$ is well-typed. Assume $\Gamma \vdash_\phi P_1|R \stackrel{s}{\Longrightarrow}$. W.l.o.g. we assume s is well-knit. By Proposition 11 we can write $\Gamma \vdash_\phi P_1|R \stackrel{s\langle u,t\rangle}{\Longrightarrow}$ with u and t legal. Hence $\Gamma \vdash_\phi P_1 \stackrel{u}{\Longrightarrow}$, that is $\Gamma \vdash_\phi P_2 \stackrel{u}{\Longrightarrow}$. Since the transitions of $P_2|R$ are solely determined by visible actions of P_2 and R we obtain $\Gamma \vdash_\phi P_2|R \stackrel{s\langle u,t\rangle}{\Longrightarrow}$, as required.
(Prefix) All rules are mechanical. For output prefixes we use the restricted form given in (8). We show the case of affine output. $\Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P_1 \sqsubseteq_{\mathsf{seq}} P_2$ s.t. $\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y})P_i$ is well-typed under $\Gamma \vdash x : \mathbf{?}_1$. Assume $\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y})P_1 \stackrel{ls}{\Longrightarrow}$ where, w.l.o.g. ls is well-knit. By the restricted form in (8), it should be (w.l.o.g.) $l = \overline{x}(\vec{y})$ and $\Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P_1 \stackrel{s}{\Longrightarrow}$. By induction hypothesis $\Gamma \cdot \vec{y} : \vec{\tau} \vdash_{\mathtt{I}} P_2 \stackrel{s}{\Longrightarrow}$, therefore $\Gamma \vdash_{\mathtt{O}} \overline{x}(\vec{y})P_2 \stackrel{ls}{\Longrightarrow}$ as required. ■

H.2 Equivalence $\approx_{\mathsf{seq}}$

Write $\approx_{\mathsf{seq}}$ for the equivalence induced by $\sqsubseteq_{\mathsf{seq}}$. We observe:

Proposition 15. $\approx_{\mathsf{seq}}$ *is a typed congruence and* $\approx_{\mathsf{seq}} \subseteq \cong_{\mathsf{seq}}$*.*

Proof: The first part is an immediate corollary of Proposition 14. The second part is by (i) and the definition of $\cong_{\mathsf{seq}}$. ■

Note Proposition 15 gives another proof of Theorem 7 (since $P \stackrel{s}{\Longrightarrow}$ and $P \longrightarrow P'$ implies $P' \stackrel{s}{\Longrightarrow}$ by determinacy).

Next we introduce a useful characterisation of $\approx_{\mathsf{seq}}$. 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. $\phi = \mathtt{I}$).

Definition 21. A family of typed relations $\{\mathbf{R}^{s_j}\}_j$ is a *sequential bisimulation* if, whenever $\Gamma \vdash_\phi P \, \mathbf{R}^s \, Q$, s matches l, and the following and its symmetric case hold: if $\Gamma \vdash_\phi P \stackrel{l}{\longrightarrow} \Delta \vdash_\psi P'$ with $s \cdot \hat{l}$ legal, then $\Gamma \vdash_\phi Q \stackrel{\hat{l}}{\Longrightarrow} \Delta \vdash_\psi Q'$ such that $\Delta \vdash_\psi P' \, \mathbf{R}^{s \cdot \hat{l}} \, Q'$. If $\Gamma \vdash_\phi P \, \mathbf{R}^s \, Q$ for some sequential bisimulation $\mathbf{R}$, we write $\Gamma \vdash_\phi P \approx^s_{\mathsf{seq}} Q$.

Proposition 16. $\approx^\varepsilon_{\mathsf{seq}} = \approx_{\mathsf{seq}}$

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

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

Here $(i = 1, 2)$. It is easy to see that $\Gamma \vdash_\phi P_1 \, \mathbf{R}^s_n \, P_2$ 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 $\Gamma \vdash_\phi P_1 \mathbf{R}^s_n P_2$ and $\Gamma \vdash_\phi P_1 \stackrel{l}{\longrightarrow} \Delta_\psi P_1'$ such that sl is legal. If $l = \tau$ we simulate this by the non-transition of $\Gamma \vdash_\phi P_2$ and the result is in $\mathbf{R}^s$ again. Otherwise this is simulated by $\Gamma \vdash_\phi P_2 \stackrel{l}{\Longrightarrow} \Delta_\psi P_2'$ and the result is in $\mathbf{R}^{sl}$ by definition, hence $\mathbf{R}_0 \subseteq \approx^\varepsilon_{\mathsf{seq}}$. ■

Proposition 17. *If $\Gamma \vdash_\phi P_1 \approx^s_{\mathsf{seq}} P_2$ and $\Gamma \vdash_\phi P_i \stackrel{l}{\longrightarrow} \Delta \vdash_\psi P_i'$ $(i = 1, 2)$ such that sl is legal, then $\Delta \vdash_\psi P_1' \approx^{sl}_{\mathsf{seq}} P_2'$.*

H.3 Context Lemma

Definition 22. (i) (reduction-closed congruence) *A* reduction-closed congruence*, say* $\mathbf{R}$*, is a typed congruence over sequential processes which satisfies the following condition: whenever $P\mathbf{R}Q$ we have (1) $P \longrightarrow P'$ implies $Q \twoheadrightarrow Q'$ such that $P'\mathbf{R}Q'$; and in addition (2) $P \downarrow_x$ implies the existence of a process Q' such that $Q \Downarrow_x$. We write $\cong'_{\mathsf{seq}}$ for the maximum reduction-closed equality.*

(ii) ($|$-congruences) *A* $|$-congruence *is a typed equivalence which includes $\equiv$ and which is closed under typed parallel composition. $\cong_{\mathsf{seq},\,|}$ denotes the maximum $|$-congruence which satisfies the same condition on observables as in $\cong_{\mathsf{seq}}$, while $\cong'_{\mathsf{seq},\,|}$ denotes the maximum $|$-congruence which satisfies the same condition as in $\cong'_{\mathsf{seq}}$.*

Proposition 18. $\cong_{\mathsf{seq}} = \cong'_{\mathsf{seq}}$. *Similarly* $\cong_{\mathsf{seq},\,|} = \cong'_{\mathsf{seq},\,|}$.

PROOF: By Proposition 15 (iii), $\twoheadrightarrow \subseteq \cong_{\mathsf{seq}}$ so that $\cong_{\mathsf{seq}}$ is reduction-closed. Since $P \cong'_{\mathsf{seq}} Q$ implies $P \cong_{\mathsf{seq}} Q$, we are done. The same reasoning applies to the latter since $\cong_{\mathsf{seq},\,|} \supseteq \cong_{\mathsf{seq}}$. ■

We can now prove the context lemma.

Lemma 23. (context lemma) $\cong_{\mathsf{seq}} = \cong_{\mathsf{seq},\,|}$.

PROOF: By Proposition 18 it suffices to show $\cong'_{\mathsf{seq}} = \cong'_{\mathsf{seq},\,|}$. By definition $\cong'_{\mathsf{seq}} \subseteq \cong'_{\mathsf{seq},\,|}$. We show the congruent closure of $\cong'_{\mathsf{seq},\,|}$ (which we write $\mathbf{R}$) is a reduction-closed congruence. For brevity we omit types in the following proof. Let $P_1 \mathbf{R} P_2$. Then $P_1 \equiv C[P_{11}]..[P_{1n}]$ and $P_2 \equiv C[P_{21}]..[P_{2n}]$ such that, for each i, $P_{1i} \cong_{\mathsf{seq},\,|} P_{2i}$.

(i) Reduction-closure. Suppose $P_1 \longrightarrow P_1'$. There are three cases.

- *The redexes are in C.* Trivial.
- *The redexes are in P_{1i}.* Immediate from $P_{1i} \cong_{\mathsf{seq},\,|} P_{2i}$.
- *One redex is in P_{1i} and another is in P_{1j} $(i \neq j)$ or in C.* We only treat the case when redexes are in P_{1i} and P_{1j}, since the other case is similar. By definition of reduction we know P_{1i} and P_{1j} are neither under input prefix nor under output prefix which binds their subjects. Noting the situation is the same for P_{2i} and P_{2j} (since input subjects are determined by action types), we can apply Fact 1 to obtain:

$$\begin{array}{l} P_1 \equiv C'[P_{1h}]_{h \notin \{i,j\}}[(P_{1i}|P_{1j})] \\ P_2 \equiv C'[P_{2h}]_{h \notin \{i,j\}}[(P_{2i}|P_{2j})]. \end{array}$$

  By assumption $P_{1i}|P_{1j} \cong_{\mathsf{seq},\,|} P_{2i}|P_{2j}$, by which P_2 simulates P_1's reduction ending up in the same closure, hence as required.

(ii) Observables. Suppose $P_1 \downarrow_x$. Then either $C \downarrow_x$ or $P_{1i} \downarrow_x$ for some i. The former case is trivial, while for the latter case we know $P_{2i} \Downarrow_x$ by $P_{1i} \cong_{\mathsf{seq},\,|} P_{2i}$, hence as required. ■

H.4 Another Characterisation of $\sqsubseteq_{\mathsf{seq}}$

The *innocent function of* $\Gamma \vdash_\phi P$, denoted $\mathsf{inn}(\Gamma \vdash_\phi P)$, is defined as the minimum function mapping visible typed traces to their one-step extensions, satisfying the condition: if $\Gamma \vdash_\phi P \stackrel{sl}{\Longrightarrow}$ and l is output, and, moreover, $\ulcorner s \urcorner^{0} = s$, then $\mathsf{inn}(\Gamma \vdash_\phi P)(s) = sl$. We always consider elements of $\mathsf{inn}(\Gamma \vdash_\phi P)$ modulo α-equality. The following characterisation is immediate from contingency completeness and determinacy of typed transitions.

Proposition 19. $\Gamma \vdash_\phi P_1 \sqsubseteq_{\mathsf{seq}} P_2$ *iff* $\mathsf{inn}(\Gamma \vdash_\phi P_1) \subseteq \mathsf{inn}(\Gamma \vdash_\phi P_2)$.

H.5 Undefined Processes

First we define $\Omega \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, xy)(!x.\overline{y}|!y.\overline{x}|\overline{x})$. Clearly $\Gamma \vdash_{\mathtt{0}} \Omega$ for an arbitrary Γ. Then Ω_x^τ (the *undefined process*) is inductively defined as follows (copy-cat agents $[x \to y]^\tau$ are given in Appendix B).

$$
\begin{aligned}
\Omega_x^{(\vec{\tau})^{!_1}} &\stackrel{\text{def}}{=} x(\vec{y}).\Omega \\
\Omega_x^{(\vec{\tau})^{!_\omega}} &\stackrel{\text{def}}{=} \begin{cases} !x(\vec{y}).\Omega_{y_i}^{\tau_i} & (\mathsf{md}(\tau_i) = ?_1) \\ !x(\vec{y}).\Omega & (\neg\exists i.\mathsf{md}(\tau_i) = ?_1) \end{cases} \\
\Omega_x^{(\vec{\tau})^{?_1}} &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z_1 z_2)([z_1 \to z_2]^{(()^{?_1})^{!_\omega}} | [z_2 \to z_1]^{(()^{?_1})^{!_\omega}} | \overline{z_1}(c)c.\overline{x}(\vec{y})\Pi_i \Omega_{y_i}^{\tau_i}) \\
\Omega_x^{(\vec{\tau})^{?_\omega}} &\stackrel{\text{def}}{=} \Omega
\end{aligned}
$$

The undefined process for branching/selection types are similarly defined. The affine channel occurring in $\Omega_x^{(\vec{\tau})^{?_1}}$ is necessary because the present type discipline demands that such a channel occurs free, though semantically it plays no role (we can indeed construct an alternative version of the sequential typing system in which we add the weakening for the affine channel and IO-mode, with exactly the same set of typed terms up to the untyped weak bisimilarity). Write $io(\tau)$ for the IO-mode corresponding to τ. We observe:

Proposition 20. *For each τ, we have $x : \tau \vdash_{io(\tau)} \Omega_x^\tau \triangleright \mathsf{md}(\tau)x$.*

To define undefined processes under general types, we use a variant of the above processes given as follows. Let τ and ρ be such that $\mathsf{md}(\tau) = !_1$ and $\mathsf{md}(\rho) = ?_1$.

$$
\Omega_{xy}^{\tau,\rho} \stackrel{\text{def}}{=} \begin{cases} x(\vec{z}).\Omega_y^\rho & (\tau \text{ unary}) \\ x[\&(\vec{z}_i).\Omega_y^\rho] & (\tau \text{ branching}) \end{cases}
$$

From Proposition 20 we easily know $x : \tau, y : \rho \vdash_{\mathtt{I}} \Omega_{xy}^{\tau,\rho} \triangleright !_1 x \otimes ?_1 y$.

Now write $\Gamma \vdash A$ when the modes given to names in A conform to Γ in the obvious sense. We then write $\Gamma \vdash_\phi A$ when $\Gamma \vdash A$ and, moreover,

(1) If $\phi = \mathtt{I}$, then the number of the channles with mode $!_1$ in A and that of those with mode $?_1$ coincide; and

(2) If $\phi = \mathtt{0}$, then either: **(2-a)** we have the same condition as **(1)** above, or **(2-b)** the number of the channles with mode $!_1$ in A is less than that of those with mode $?_1$ coincide exactly by one.

We can now construct the undefined processes for general types as follows. We consider only those cases when action types do not contain $\perp$ since such channels can simply be restricted.

Definition 23. *Let $\Gamma \vdash_\phi A$ such that $\perp \notin \mathsf{md}(A)$. Then we define $\Omega_{A,\phi}^\Gamma$ as follows.*

(i) *If $\phi = \mathtt{I}$ and hence the condition* **(1)** *above is satisfied, then for $\vec{y}$ of mode $!_1$ and $\vec{x}$ of mode $?_1$ in A, together with $\vec{w}$ of mode $!_\omega$ in A, we define $\Omega_{A,\phi}^\Gamma \stackrel{def}{=} (\Pi_i \Omega_{x_i y_i})|(\Pi_j \Omega_{w_j})$.*

(ii-a) *If $\phi = \mathsf{o}$ and the condition* **(2-a)** *above is satisfied, then, letting $\vec{y}$, $\vec{x}$ and $\vec{w}$ be as* **(i)** *above,* $\Omega^{\Gamma}_{A,\phi} \stackrel{def}{=} (\Pi_i \Omega_{x_i y_i})|(\Pi_j \Omega_{w_j})|\Omega$.

(ii-b) *If $\phi = \mathsf{o}$ and the condition* **(2-a)** *above is satisfied, then, letting $\vec{y}$, $\vec{x}x'$ and $\vec{w}$ as* **(i)** *above assuming $\vec{x}$ and $\vec{y}$ are in bijection, we set* $\Omega^{\Gamma}_{A,\phi} \stackrel{def}{=} (\Pi_i \Omega_{x_i y_i})|(\Pi_j \Omega_{w_j})|\Omega_{x'}$.

It is easy to check $\Gamma \vdash_\phi \Omega^{\Gamma}_{A,\phi} \triangleright A$. Before showing $\Omega^{\Gamma}_{A,\phi}$ is indeed the least defined process in each type, we first observe:

Proposition 21. *$\Gamma \vdash_\phi A$ iff for some P we have $\Gamma \vdash_\phi P \triangleright A$.*

Proof: The "if" direction is by induction on typing rules. For (Inact), the condition **(1)** trivially holds. (Par), (Res), the output prefix rules and the weakening rules are also trivial. For (In$^{!_1}$), we assume the condition **(2)** (either of **(2-a)** and **(2-b)** is fine) to hold in the premise by the induction hypothesis, and cut off several $?_\omega$-nodes from the action type, and add one $!_1$, so that the condition **(1)** does hold in the consequence. For (In$^{!_\omega}$), in the premise we should have the degenerate case of **(2)** (with no affine input) in the action type, so that **(1)** easily holds in the consequence, hence done. For the "only if" direction, we simply use $\Omega^{\Gamma}_{A,\phi}$ if there is no $\perp$ in A. If there is, let $A = B \otimes \perp\vec{u}$. We now define, with $\mathsf{md}(\tau) = !_1$, $\mathbf{L}^{\tau}_x \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, y)(x(\vec{w}).\overline{y}|y.\Omega^{\overline{\tau}}_x)$. Assuming the input part of the types of $\vec{u}$ being $\vec{\tau}$, we can now take, the process $\Omega^{\Gamma}_{B,\phi}|\Pi_i \mathbf{L}^{\tau_i}_{u_i}$, which does type-check under Γ, ϕ and A, as required. ■

We can finally reach the main result of this subsection.

Proposition 22. (undefined processes)

(i) $x : \tau \vdash_{io(\tau)} \Omega^{\tau}_x \sqsubseteq_{\mathsf{seq}} P \triangleright \mathsf{md}(\tau)x$ for any $x : \tau \vdash_{io(\tau)} P \triangleright \mathsf{md}(\tau)x$.

(ii) Let $\Gamma \vdash_\phi A$ such that $\perp \notin \mathsf{md}(A)$. Then $\Gamma \vdash_\psi \Omega^{\Gamma}_{A,\psi} \sqsubseteq_{\mathsf{seq}} P$ for any $\Gamma \vdash_\psi P \triangleright A$.

Proof: For both (i) and (ii), we merely note that these agents have no visible output after the the obligatory inputs (if any). ■

H.6 Tester Lemma

Definition 24. Let A, B be action types. We write $A \perp B$ when the following and its symmetric case holds: if $px \in A$ and $p \notin \{\perp\}$ then $\overline{p}x \in B$.

Below we say A is *closed* if $px \in A$ implies $p \in \{\perp, !_\omega\}$.

Proposition 23. *$A \perp B$ implies (1) $A \asymp B$ and (2) $A \odot B$ is closed.*

Lemma 24. **(Lid Lemma)** *Let $x : \tau \cdot y : \rho \cdot \Gamma \vdash_{\mathsf{o}} P \triangleright A \otimes px \otimes ?_1 y$ with $io(\rho) = \mathsf{o}$. Then (i) $y : \rho \cdot \Gamma \vdash_{\mathsf{o}} P|\Omega^{\overline{\tau}}_x \triangleright A \otimes ?_1 y$ and (ii) $P \Downarrow_y$ iff $P|\Omega^{\overline{\tau}}_x \Downarrow_y$.*

PROOF: Both are immediate from the construction. ■

We say Γ is *saturated* when each type mentioned in Γ is paired. Note whenever $\Gamma \vdash_\phi P \triangleright A$ we can saturate Γ without changing typability.

Lemma 25. **(Tester Lemma)** *Assume* Γ *is saturated. Then* $\Gamma \vdash_{\mathtt{I}} P_1 \not\cong_{\mathsf{seq}} P_2 \triangleright A$ *iff there exists* $\Gamma' \vdash_{\mathtt{O}} R \triangleright B \otimes ?_1 x$ *with* $\Gamma' = x : ()^{?_1} \cdot \Gamma$ *such that (1)* $B \bot A$ *and (2)* $\Gamma' \vdash P_1 | R \Downarrow_x$ *and* $\Gamma' \vdash P_2 | R \Uparrow$*, or its symmetric case.*

PROOF: By Lemma 23, there exists $\Gamma \cdot \Delta \cdot z : \tau \vdash_{\mathtt{O}} S$ with $\mathsf{md}(\tau) = ?_1$ such that, say, $(P_1|S) \Downarrow_z$ and $(P_2|S) \Uparrow$, for which we again assume Δ is saturated without loss of generality. Assume

$$\Gamma \cdot \Delta \cdot z : \tau \vdash_{\mathtt{O}} S \triangleright C \otimes C' \otimes ?_1 z$$

where C does not contain a node of form $!_1 w_i$. Then we can write $S \equiv T|T'$ where T' are the composition of affine input processes whose active subjects are from C', so that we have

$$\Gamma \cdot \Delta \cdot z : \tau \vdash_{\mathtt{O}} T \triangleright C \otimes ?_1 z.$$

Since the processes in T' cannot be engaged in interaction, we still have $(P_1|T) \Downarrow_z$ and $(P_2|T) \Uparrow$. Let $S \stackrel{\text{def}}{=} z(\vec{u}).\overline{x}|T$. We now convert S into the stated form, by closing Δ by restriction. Let $C = C_0 \otimes C'$ such that $\Gamma \vdash C_0$ and $\Delta \vdash C'$, such that $C' = (\otimes_i p_i y_i) \otimes !_\omega C''$. By construction we know $p_i \in \{?_\omega, ?_1\}$. Let $R_0 \stackrel{\text{def}}{=} S|\Pi\Omega^{\rho_i}_{y_i}$ where $\Gamma \vdash y_i : \rho_i$ and $\mathsf{md}(\rho_i) = \overline{p_i}$ (i.e. ρ_i is the compensating input type). Then we have

$$\Gamma \cdot \Delta \cdot z : \tau \cdot x : ()^{?_1} \vdash_{\mathtt{O}} R_0 \triangleright C_0 \otimes \bot E \otimes !_\omega F \otimes ?_1 x.$$

By Lemma 24, we know $P_i|R_0 \Downarrow_x$ iff $P_i|S \Downarrow_z$. By taking $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \vec{y})R_0$ we are done. ■

I Finite Testability

This section introduces an abstract notion of innocent function (which does not use the term structure) and show that any finite such function is realisable by a sequential process.

I.1 Innocent Functions under Type Γ, A, ϕ

Let $\Gamma \vdash_\phi A$. Then the transition relation of form $\Gamma \vdash_\phi A \xrightarrow{l} \Delta \vdash_\psi B$ $(l \neq \tau)$ is generated from the following rules (we show the case of branching: the unary cases are given accordingly).

$$(\text{IN}) \quad \frac{\Gamma \vdash l : [\&_{i\in I}\vec{\tau}_i]^p \quad A \vdash px \quad \mathsf{md}(\vec{\tau}_j) = \vec{q} \ \ (j \in I)}{\Gamma \vdash_{\mathtt{I}} A \stackrel{x\mathtt{in}_j(\vec{y})}{\longrightarrow} \Gamma \cdot \vec{y} : \vec{\tau}_j \vdash_{\mathtt{O}} A \otimes \vec{q}\vec{y}}$$

$$(\text{OUT}) \quad \frac{\Gamma \vdash l : [\oplus_{i\in I}\vec{\tau}_i]^p \quad A \vdash px \quad \mathsf{md}(\vec{\tau}_j) = \vec{q} \ \ (j \in I)}{\Gamma \vdash_{\mathtt{O}} A \stackrel{\overline{x}\mathtt{in}_j(\vec{y})}{\longrightarrow} \Gamma \cdot \vec{y} : \vec{\tau}_j \vdash_{\mathtt{I}} A \otimes \vec{q}\vec{y}}$$

We can check whenever $\Gamma \vdash_\phi A$ and $\Gamma \vdash_\phi A \xrightarrow{l} \Gamma \vdash_\psi B$ we have $\Gamma \vdash_\psi B$, so that the transition is well-defined. We write $\Gamma \vdash_\phi A \xrightarrow{s} \Delta \vdash_\psi B$ for a transition sequence. Such s is called *sequence under* Γ, ϕ, A. Such a sequence is *legal* (resp. *well-knit*, resp. an *output-view*) when it satisfies the same conditions as Section 4.3 (we call a transition sequence an *output-view* if it is the output view of another legal transition sequence: in particular, such a sequence is the output-view of itself). Two non-empty well-knit sequences s_1 and s_2 under Γ, ϕ, A are *coherent* if, whenever their initial free subjects differ, their affine output subjects are disjoint. We can now define innocent function without mentioning concrete terms.

Definition 25. Let $\Gamma \vdash_\phi A$. Then an *innocent function under* Γ, ϕ, A is a prefix-closed set f of mutually coherent output views under Γ, ϕ, A satisfying the following conditions (sequences are considered modulo $\equiv_\alpha$):

(i) (contingency completeness) If $s \in f$, $\Gamma \vdash_\phi A \xrightarrow{s} \Delta \vdash_{\mathtt{I}} B \xrightarrow{l}$ and sl is an output view, then $sl \in f$.
(ii) (innocence) If $sl_i \in f$ with l_i output $(i = 1, 2)$ then $l_1 = l_2$.

We write $f : \langle \Gamma, A, \phi \rangle$ when f is an innocent function under Γ, A, ϕ. An innocent function f is *finite* when its cardinality modulo $\equiv_\alpha$, denoted $|f|$, is finite.

An innocent function in the above sense indeed defines a function by simply taking as its range a set of sequences ending with output (thus the finiteness in the above sense corresponds to finiteness in terms of the graph of a function). For this reason we often confuse such f with the induced function. Since names with modes $\perp$ in an action type never induces transition, we can safely stipulate:

Convention 2 *From now on whenever we consider an innocent function* $f : \langle \Gamma, \psi, A \rangle$, *we assume* $px \in A$ *implies* $p \neq \perp$.

I.2 Representability

Our aim in this section is to show that any finite innocent function is representable by concrete terms. We use the following two lemmas, one about the innocent function in the above sense, another about legal sequences.

Lemma 26. *Let* $f : \langle \Gamma, A, \mathtt{I} \rangle$ *with* $A = ?_\omega B \otimes ?_1 C \otimes p_1 x_1 \otimes .. \otimes p_n x_n$. *Then* $f = \uplus f_i$ *with* $f_i : \langle \Gamma, A_i, \mathtt{I} \rangle$ *where* $A_i = ?_\omega B \otimes ?_1 C_i \otimes p_i x_i$ *and* $\uplus C_i = C$.

Proof: Since sequences in f are well-knit, f can be partitioned into f_i, each containing all and only sequences in f starting from an action with subject x_i. By coherence and well-knitness, $\mathsf{fn}(s_i) \cap \mathsf{fn}(s_j) \subseteq \mathsf{fn}(B)$ whenever $s_i \in f_i$ and $s_j \in f_j$ with $i \neq j$. Thus $\Gamma \vdash_{\mathtt{I}} A_i \xrightarrow{s_i}$ for each $s_i \in f_i$. Clearly f_i satisfies contingency completeness and innocence under this new type, hence done. ∎

Lemma 27. *Let* $\Gamma \vdash_{\mathtt{I}} A \xrightarrow{ls}$ *such that (1)* ls *is an output-view and (2)* $\Gamma \vdash l : !_\omega$. *Then no free affine output occurs in* s.

PROOF: By contradiction. We use $[, (,),]$ for $!_\omega, ?_\omega, !_1, ?_1$ for the ease of understanding. Suppose $s = [_1 t]_j u$ such that $]$ is free. By well-bracketing $[_1$ should be answered by an action in t. Let this action be $]_i$ (hence $i \curvearrowright_{\mathsf{b}} j$), so that $t = t_0]_j t_1$. Since s is an output view, the initial action of t_1 should be bound by $]_j$. We show that this cannot be the case. Initially the last action of t_1 can only be input hence). Since s is an output view this should be bound by the previous action, hence we know $t = t'()$. Assume we already know $t = t''()..()$. Since $t]_j$ is well-bracketed and $]_j$ is free, again the last action of t'' should be), hence the previous action, either in t'' or $]_j$ itself, can only be (, contradiction. ∎

Theorem 9. (representability) *Let f be a finite innocent function under Γ, ϕ, A. Then there exists $\Gamma \vdash_\phi P \triangleright A$ such that* $\mathsf{inn}(\Gamma \vdash P) = f$.

PROOF: We use induction on the cardinality of finite innocent functions under all types, cf. [28]. Sequences are always considered modulo $\equiv_\alpha$, and we assume that in (the sequences in) $f : \langle \Gamma, \phi, A\rangle$ all names in A occur free (for input this is vacuous; for output this does not lose generality since once we obtain a process which does interact at $A_0 \subseteq A$, then we can simply add an appropriate set of $\mathbf{L}$ to add $A \backslash A_0$ to the action type). For the base case, we have $|f| = 0$, in which case $\Omega^\Gamma_{A,\psi}$ satisfies the condition under an arbitrary type. For the inductive step, assume that the statement holds for all innocent functions with cardinality no more than n. Assume in addition that f is an innocent function under Γ, ϕ, A and $|f| = n + 1$. We first consider the case when $\phi = \textsc{i}$. By Lemma 26, if we write $A = ?_\omega B \otimes ?_1 C \otimes p\vec{x}$ where each p_i is $!_1$ or $!_\omega$, we can write $f = \uplus f_i$ with $f_i : \Gamma, A_i, \textsc{i}$ where $A_i = ?_\omega B \otimes ?_1 C_i \otimes p_i x_i$ where $C = \uplus C_i$. If $|f_i| \subsetneq |f|$ for each i, then by induction hypothesis there exists $\Gamma \vdash_{\mathtt{I}} P_i \triangleright A_i$ which represents f_i for each i. Take $\Gamma \vdash_{\mathtt{I}} \Pi_i P_i \triangleright A$ (which is well-typed by the conditions on A_i) which clearly represents f. So suppose $|f_i| = |f|$ for some i, so that $|f_j| = 0$ for $i \neq j$. We only have to consider the representation of f_i. There are two cases.

Case $p_i = !_1$. We treat the case when $\Gamma \vdash x : [\&_{j\in J}(\vec{\tau}_j)]^{!_1}$ since this subsumes the unary case. By contingency completeness, for each $j \in J$, there is a family of sequences each of form $x\mathtt{in}_j(\vec{y}_j)s_j$. This family, say g_j, defines an innocent function since the initial output is identical in all of them (by innocence). By induction hypothesis, g_j is represented by a well-typed $\Gamma \cdot \vec{y}_j : \vec{\tau}_j \vdash_{\mathtt{O}} Q_j \triangleright A_i \otimes E_j$, where E_j assigns appropriate modes to $\vec{y}_j$. We can now take $P_i \stackrel{\text{def}}{=} x[\&_j(\vec{y}_j).Q_j]$. Clearly we have $\Gamma \vdash_{\mathtt{I}} P_i \triangleright A_i$, which represents f_i, hence done.

Case $p_i = !_\omega$. We again only treat the case when $\Gamma \vdash x : [\&_{j\in J}(\vec{\tau}_j)]^{!_\omega}$. By Lemma 27, if $s \in f_i$ then s does not contain a free affine output, hence neither does A_i by assumption. As before, for each $j \in J$, there is an innocent function g_j which is represented by, say, $\Gamma \vdash_{\mathtt{O}} Q_j \triangleright A_i \otimes E_j$ for an appropriate E_j. Since each A_i does not contain a free affine output, $\Gamma \vdash_{\mathtt{I}} !x_i[\&_{j\in J}(\vec{y}_j).P_j] \triangleright A_i$ is well-typed which has the expected behaviour, hence done.

Finally the case when $\phi = \textsc{o}$ is obvious since, after the unique initial output, the remaining sequences define an innocent function of strictly less cardinality with mode $\textsc{i}$. ∎

An important corollary of Theorem 9 follows.

Proposition 24. *Let* $\Gamma \vdash_{\mathtt{O}} P|R \Downarrow_x$. *Then for some finite* $\Gamma \vdash_\phi R_0 \sqsubseteq_{\mathsf{seq}} R$ *we have* $\Gamma \vdash_{\mathtt{O}} P|R_0 \Downarrow_x$.

PROOF: Since $\Gamma \vdash_{\mathtt{O}} P|R \Downarrow_x$ we observe $\Gamma \vdash_{\mathtt{O}} P|R \stackrel{l}{\Longrightarrow}$ such that $\mathsf{sbj}(l) = x$, which can be written $\Gamma \vdash_{\mathtt{O}} P|R \stackrel{l\langle s,t\rangle}{\Longrightarrow}$. We then take R_0 as the process which represents the minimum innocent function which contains prefixes of t (which is finite). ■

Note that if $\Gamma \vdash_{\mathtt{O}} P|R \Uparrow$ then immediately $\Gamma \vdash_{\mathtt{O}} P|R_0 \Uparrow$ too (since $\sqsubseteq_{\mathsf{seq}}$ is a precongruence).

The proof of Proposition 7 is by evident bisimulations. We move to the proof of Proposition 8.

I.3 Proof of Proposition 8 (Finite Testability)

By Lemma 25 (Tester Lemma), we can find some R' such that, for $\Delta \stackrel{\text{def}}{=} \Gamma \odot \overline{\Gamma}$ and $\Delta \cdot x : ()^{?_1} \vdash_{\mathtt{O}} R' \triangleright B$ where $B = \otimes_i \mathbf{!}_\omega x_i \otimes \mathbf{?}_1 x$ which differentiates P_1 and P_2 when composed in parallel. By construction we can write $R' \equiv (\boldsymbol{\nu}\, \vec{z})(\Pi R_i\langle x_i\rangle \,|\, \Pi_j R'_j\langle z_j\rangle \,|\, S'\langle zx\vec{x}\rangle)$ where $R_i\langle x_i\rangle$ (resp. $R'_j\langle z_j\rangle$) is a replicated term with subject x_i (resp. z_j), both of which may access x_i and z_j, while S' is a process with mode $\mathtt{o}$ and which may access z, x, x_i and z_j (note a replicated input cannot contain free affine channels so only S' can access z and x). We then use Proposition 7 to make a private copy for each replicated process which a process uses at channels of mode $\mathbf{?}_\omega$, so that R' is transformed into:

$$\Delta \cdot x : ()^{?_1} \vdash_{\mathtt{O}} R' \approx_{\mathsf{seq}} \Pi_i R_i\langle x_i\rangle \mid S\langle zx\rangle$$

where $\mathsf{fn}(S) = \{zx\}$ and $R_i\langle x_i\rangle$ is a replication with subject x_i which should be its only free name. We now apply Proposition 24 to replace each R_i and S with the corresponding finite processes with the same effect, hence as required. ■

Remark. The final part uses the order-preserving nature of $\sqsubseteq_{\mathsf{seq}}$ and representability of finite innocent functions as processes. This part can also be proved using the correspondence with strategies. Let $\theta \stackrel{\text{def}}{=} (()^{?_1})^{!_\omega}$ and let $r : \rho_1 \times \cdots \times \rho_n$, $p_i : (\rho_1 \times \cdots \times \rho_n) \to \theta$ and $s : \tau \Rightarrow \theta$ be innocent strategies corresponding to R, P_i and S, respectively (here by abuse of notation we use channel types to denote the corresponding arenas). Then $s \circ p_i \circ r : \theta$ represents $w : \theta \vdash_{\mathtt{I}} (\boldsymbol{\nu}\, \vec{x}u)(R|P_i|!w(v).S)$ for $i = 1, 2$, i.e.

$$\Delta \cdot u : \overline{\tau} \cdot v : ()^{?_1} \vdash_{\mathtt{O}} (R|P_i|S) \Downarrow_v \quad \Leftrightarrow \quad s \circ p_i \circ r \neq \bot : \theta$$

where $\bot$ is the bottom element. By the CPO structure of homsets and continuity of composition in the category of games, we can take finite $s_0 \subseteq s$ and $r_0 \subseteq r$ with the same property. These strategies can in turn be considered as transitions, which are then representable as processes. Note that the pair R and S (with replication for the latter) precisely corresponds to the pair of testing morphisms used by [3] and [28] in their PCF games.

J Full Abstraction

This section first establishes computational adequacy, then proves definability of finite processes as PCF-terms. For computational adequacy we show a simplest proof in outline, using operational correspondence, cf. [34]. For definability we use the correspondence between finite canonical normal forms of PCF terms, on the one hand, and "normal forms" of typable processes, on the other.

J.1 Computational Adequacy

We outline the proof of computational adequacy via operational correspondence. For the purpose it is convenient to use an extension of $\longrightarrow$, which we write $\mapsto$, called *extended reduction*, first introduced in [45]. The extended reduction is defined as the compatible closure generated from the following rules over processes modulo $\equiv$, assuming the standard variable convention for bound names.

$$\begin{array}{rll} x(\vec{y}).Q|C[\overline{x}(\vec{y})P] \mapsto C[(\boldsymbol{\nu}\,\vec{y})(P|Q)] & & (\mathsf{fn}(Q)\cap\mathsf{fn}(C[\;])=\emptyset) \\ !x(\vec{y}).Q|C[\overline{x}(\vec{y})P] \mapsto C[(\boldsymbol{\nu}\,\vec{y})(P|Q)]|!x(\vec{y}).Q & & (\mathsf{fn}(Q)\cap\mathsf{fn}(C[\;])=\emptyset) \\ (\boldsymbol{\nu}\,x)!x(\vec{y}).P \mapsto \mathbf{0} & & \end{array}$$

Immediately $\longrightarrow\subset\mapsto$. Further we can easily check $\mapsto$ is a bisimulation, hence $\mapsto\subset\cong_{\mathsf{seq}}$. We can further verify the following. In (ii) below we write $P \Uparrow_e$ when $\forall n \in \omega.\ P \mapsto^n$. In (iii) Ω_u appeared in Appendix H.5.

Proposition 25. (i) *Let* $\vdash M : \mathsf{Nat}$. *Then* $M \longrightarrow M'$ *implies* $[\![M : \mathsf{Nat}]\!]_u \mapsto^+ [\![M' : \mathsf{Nat}]\!]_u$.

(ii) *If* $[\![M : \mathsf{Nat}]\!]_u \mapsto^* P' \not\mapsto$, *then* $P' \equiv [\![n : \mathsf{Nat}]\!]_u$ *for some* n.

(iii) *If* $[\![M : \mathsf{Nat}]\!]_u \Uparrow_e$ *then* $[\![M : \mathsf{Nat}]\!]_u \approx \Omega_u$.

Proof. (i) is mechanical. For (ii), first $P' \equiv !u(c)R$ by typability. Since (1) R is in the output mode (2) c occurs uniquely in R and (3) no other free output channel is in R either it contains a hidden redex or c is active. By $R \not\mapsto$ only the latter is possible, hence $R \equiv \overline{c}\mathtt{in}_n R'$. By typing rule we can easily check $R' \equiv \mathbf{0}$,hence done. This also proves (iii). ∎

Combining Proposition 25 (i) and (ii) we are done.

J.2 Finite Definability (1): Ω-normal Forms

The proof of full abstraction relies on PCF-definability of finite processes (Theorem 2). This can be proved in several ways. We can use the definability result of games in [3, 28], via the correspondence between composition of strategies and composition of processes; alternatively, we can work directly with processes, using induction on the cardinality of (finite) innocent functions under PCF types (in the sense of Appendix I.1). A more direct method is to can use a syntactic characterisation of finite processes, extending $\mapsto$ (introduced in Appendix J.1) to $\mapsto_\Omega$ as follows. Below Ω_u appeared in Appendix G.5. In the second rule $\Uparrow_e$ is the divergence w.r.t. the extended reduction. In the last rule we assume the typability.

$$\frac{\Gamma \vdash_\phi P \mapsto P'}{\Gamma \vdash_\phi P \mapsto_\Omega P'} \qquad \frac{\Gamma \vdash_\phi P \Uparrow_e}{\Gamma \vdash P \mapsto_\Omega \Omega^\Gamma_{A,\phi}} \qquad \frac{\Gamma \vdash P \mapsto_\Omega Q}{\Delta \vdash C[P] \mapsto_\Omega C[Q]}$$

It is easy to check $\mapsto_\Omega$ stays within the weak bisimilarity introduced in Section 4. We say a typable process P is in *Ω-normal form* if P has no reduction except within Ω^τ_x (in other words if P convergences assuming the undefined processes are constants for their respective types). The following inductive construction of Ω-normal forms is notable.

Definition 26. *The set of* syntactic Ω-normal forms *is inductively generated by the following rules. We implicitly assume typability.*

- $\mathbf{0}$ *is a Ω-normal form.*
- *Each $\Omega^\Gamma_{A,\phi}$ is a Ω-normal form.*
- *If P is a Ω-normal form, then $x(\vec{y}).P$ is a Ω-normal form. Similarly for branching.*
- *If P is a Ω-normal form, then $!x(\vec{y}).P$ is a Ω-normal form.*
- *If P is a Ω-normal form such that all of its free active output subjects are within $\vec{y}$, then $\overline{x}(\vec{y})P$ is a Ω-normal form, similarly for selection.*
- *If each P_i is a Ω-normal form such that none share compensating input and output, then ΠP_i is a Ω-normal form.*

Immediately each syntactic Ω-normal form is indeed a Ω-normal form. Further we observe the fundamental result.

Proposition 26. *Let f be a finite innocent function under Γ, A, ϕ. Then there exists a Ω-normal form P generated inductively as above such that $\Gamma \vdash_\phi P \triangleright A$ and* $\mathsf{inn}(P) = f$.

Proof. By syntactically following the proof of Theorem 9, observing each construction of processes in the proof follow the induction given above. ■

In fact, it is easy to check that the innocent function of each syntactic Ω-normal form is always essentially finite in that it is finite except it may be defined for an infinite number of branches in some branching types (i.e. its depth is always finite though its width may be infinite), even though we do not use this property below.

J.3 Finite Definability (2) Finite Canonical Forms

The following set of terms is the restriction of the preceding inductive construction of Ω-normal forms to those typable under the (translation of) PCF-types, which we call *PCF Ω-normal forms.* We again implicitly assume typability.

(a) Ω_u and $!u(c)\overline{c}\mathtt{in}_n$ are PCF-Ω-normal forms.
(b) If each of $\{P_i\}$ $(i \in I)$ and $\{!u(c).Q_j\}$ $(j \in \omega)$ is a PCF-Ω-normal form, then $!u(c).\overline{x}(\vec{y}e)(\Pi P_i\langle y_i\rangle \mid e[\&_{j\in\omega} Q_j]$ is a PCF Ω-normal form.
(c) If $!u(\vec{y}c).P$ is a PCF Ω-normal form then $!u(x\vec{y}c).P$ is a PCF Ω-normal form.

A PCF Ω-normal form is *finite* if it is constructed by the same rules except, in (b), we set that $!u(c)Q_j \equiv \Omega_u$ for co-finitely many j. It is easy to check all PCF Ω-normal forms are indeed Ω-normal forms.

The syntactic form introduced thus closely corresponds to *finite canonical forms* (FCFs), introduced in [3, 28], which are given as follows.

$$F \ ::= \ \Omega \mid n \mid \lambda x : \alpha.F \mid \mathsf{case}\ x\vec{F}\ \mathsf{of}\ \{n_j : F'_j\}_{j\in J}$$

where, in the last line, we assume I to be a finite non-empty subset of $\mathbb{N}$ and that each of $x\vec{F}$ and F'_i has type Nat. Finite PCF Ω-normal forms are translated into FCFs as follows.

(a') $(!u(c).\overline{c}\mathtt{in}_n)^\star \stackrel{\text{def}}{=} n$ and $(!u(c).\Omega_c)^\star \stackrel{\text{def}}{=} \Omega$.
(b') If $P_i^\star \stackrel{\text{def}}{=} F_i$ $(i \in I)$ and $(!u(c).Q_j)^\star \stackrel{\text{def}}{=} F'_j$ $(j \in \omega)$ with $!u(c).Q_j \equiv \Omega_u$ for cofinitely many j, then

$$(!u(c).\overline{x}(\vec{y}e)(\Pi P_i\langle y_i\rangle \mid e[\&_{j\in\omega} Q_j])^\star \ \stackrel{\text{def}}{=}\ \mathsf{case}\ x\vec{F}\ \mathsf{of}\ \{n_j : F'_j\}_{j\in J}$$

(c') If $(!u(\vec{y}c).P)^\star \stackrel{\text{def}}{=} F$ then $(!u(x\vec{y}c).P)^\circ \stackrel{\text{def}}{=} \lambda x.F$.

Observe that $P^\star$ is bijective, so that its inverse gives a translation of FCFs into (PCF) Ω-normal forms. Writing this translation as F°, immediately we have:

Proposition 27. *Let P be an Ω-normal form. Then $(P^\star)^\circ \approx P$.*

The finite PCF Ω-normal forms can also be mapped into typable processes by first translating them into (geneuine) PCF terms and then using the map $[\![\cdot]\!]_u$. For simplicity we use the syntax $\mathtt{if}\ M_1 = M_2\ \mathtt{then}\ N\ \mathtt{else}\ L$, which is easily translatable into the syntax given in Section 5.1, for which we assume the standard operational semantics and the translation into π-terms (the subsequent arguments do not essentially differ if we directly translate the case construct into the original syntax).

$$\begin{aligned}
(\Omega : \alpha)^\diamond &\stackrel{\text{def}}{=} \mu x : \alpha.x\\
(\lambda x : \alpha.F)^\diamond &\stackrel{\text{def}}{=} \lambda x : \alpha.F^\diamond\\
(\mathsf{case}\ xF_1..F_m\ \mathsf{of}\ \{n_i : F'_i\}_{i\in I})^\diamond &\stackrel{\text{def}}{=} \mathtt{if}\ N = n_1\ \mathtt{then}\ F'_1\ \mathtt{else}\\
&\quad\ \mathtt{if}\ N = n_2\ \mathtt{then}\ F'_2\ \mathtt{else}\\
&\quad\ \ldots\\
&\quad\ \mathtt{if}\ N = n_m\ \mathtt{then}\ F'_m\ \mathtt{else}\Omega
\end{aligned}$$

where, in the third line, we set $I = \{1, .., m\}$ and $N \stackrel{\text{def}}{=} xF_1^\diamond..F_m^\diamond$. We now observe:

Proposition 28. *Let F be a FCF. Then $F^\circ \cong_{\mathsf{seq}} [\![F^\diamond]\!]_u$.*

Proof. By induction on the construction of FCFs. The only non-trivial case is the translation of the case construct, which we treat below. For simplicity we assume the body of the case construct is defined only for $j = 0, 1$. We write $e[Q_0, Q_1; R]$ for $e[\& Q_i]$ such that $Q_i \stackrel{\text{def}}{=} R$ for $i \notin \{0, 1\}$. Then we have to show, with $Q_3 = \Omega_c$,

$$S \stackrel{\text{def}}{=} !u(c).\overline{x}(\vec{y}e)(\Pi P_i\langle y_i\rangle \mid e[Q_0, Q_1; Q_2]$$

and

$$T \stackrel{\text{def}}{=} !u(c).\overline{x}(\vec{y}e)(\Pi P_i\langle y_i\rangle \mid e[Q_0, (\overline{x}(\vec{y}e')(\Pi P_i\langle y_i\rangle \mid e'[Q_2, Q_1; Q_2])); Q_2]$$

are equal up to $\cong_{\mathsf{seq}}$. But by Lemma 25, we have only to take their observables in the context in which all free $\mathbf{?}_\omega$-names are compensated by some processes. Note that, given such a context (say $C[\,\cdot\,]$), the value e receives in S and the values e and e' receive in T always coincide. Thus we have (assuming compensating names are restricted in the context):

$$C[T] \mapsto^* !u(c)Q_i \text{ iff } C[S] \mapsto^* !u(c)Q_i | R$$

by which we conclude $T \cong_{\mathsf{seq}} S$, as required. ■

We can now establish Theorem 2. For reference we restate the theorem below.

Theorem 10. (finite definability) *Let $E^\circ \cdot u : \alpha^\circ \vdash_{\mathtt{I}} P \triangleright \mathbf{!}_\omega u$ be finite. Then $E^\circ \cdot u : \alpha^\circ \vdash [\![M : \alpha]\!]_u \cong_{\mathsf{seq}} P$ for some M.*

Proof. If $E^\circ \cdot u : \alpha^\circ \vdash_{\mathtt{I}} P \triangleright \mathbf{!}_\omega u$ is finite, then its innocent function can be represented by a (syntactic) Ω-normal form, say P_0. Since their innocent functions are identical, we have $P_0 \approx P$. Take $M \stackrel{\text{def}}{=} (P_0^\circ)^\diamond$. Then by Proposition 27 and 28, we have $[\![M]\!]_u \cong_{\mathsf{seq}} P_0 \approx P$, as required. ■