# Strong Normalisation in the $\pi$-Calculus

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

[†] *Department of Computing, Imperial College London*

[‡] *Department of Computer Science, Queen Mary, University of London*

We introduce a typed $\pi$-calculus where strong normalisation is ensured by typability. Strong normalisation is a useful property in many computational contexts, including distributed systems. In spite of its simplicity, our type discipline captures a wide class of converging name-passing interactive behaviour. The proof of strong normalisability combines methods from typed $\lambda$-calculi and linear logic with process-theoretic reasoning. It is adaptable to systems involving state, nondeterminism, polymorphism and other extensions. Strong normalisation is shown to have significant consequences, including finite axiomatisation of weak bisimilarity, a fully abstract embedding of the simply-typed $\lambda$-calculus with products and sums and basic liveness in interaction. Strong normalisability has been extensively studied as a fundamental property in functional calculi, term rewriting and logical systems. This work is one of the first steps to extend theories and proof methods for strong normalisability to the context of name-passing processes.

*Key Words:* concurrency theory, the $\pi$-calculus, strong normalisation, type theory

## 1. INTRODUCTION

**Background.** The formal study of types in programming languages and computational calculi has led to the understanding that types can ensure a wide range of desirable computational properties, ranging from error-free execution to precise logical specification of program behaviour. One important property in this context, widely found in typed $\lambda$-calculi, is *strong normalisation* (SN), which says that computation in programs necessarily terminates regardless of evaluation strategy. This is interesting from a logical viewpoint especially because, by the correspondence between proofs and programs, SN of certain $\lambda$-calculi implies consistency of the corresponding logical systems. For this reason, functional calculi and logics have been the main focus in the study of strong normalisability.

The significance of SN is, however, not limited to this traditional setting. SN is also interesting in the context of communicating processes. As an example, consider a distributed client-server interaction: when a client requests some service, s/he may naturally wish the computation on the server's side to terminate and return an answer. SN is thus a basic requirement for, say, interaction between banks and their customers. As another example, the resource preservation guaranteed by SN has been one of the main reasons for Gunter and his colleagues to develop their typed programming language for active networks (PLAN)

[26, 56] on the basis of a simply typed λ-calculus. Such languages would in general require primitives for communication and concurrency. This suggests a systematic effort to extend the accumulated theories of functional SN types to the realm of interactivity is a worthwhile endeavour.

We are thus motivated to reposition and study strong normalisability in the context of process theory. In particular, is there a basic typed process calculus in which strongly normalising functional calculi are faithfully embeddable? By faithful, we mean that typability of the encoding automatically ensures strong normalisability of the source calculus. More ambitiously, can we obtain exact semantic correspondence, including full abstraction and full completeness? Obtaining affirmative answers to these questions would not be of mere theoretical interest: since typed λ-calculi offer a basic theory of procedure calls, a fundamental abstraction in programming, embeddability of SN functional calculi would capture interactive behaviour powerful enough to involve non-trivial procedural calls while maintaining SN. Exploration of strong normalisability in this broader context might also shed new light on typed functional computation itself.

The present work is a trial in this direction, introducing a typed π-calculus in which first-order strongly normalising λ-calculi are fully abstractly embeddable. The type discipline simply adds causal chains to the system introduced in [11] where we established fully abstract encodings of PCF/PCFv [25]. This small addition radically changes the class of typable process behaviour, turning possibly diverging computation into a strongly normalising one. As would be imagined by the embeddability of typed λ-calculi, the proof of SN is non-trivial, defying naive structural induction. We adapt methods developed for strongly normalising λ-calculi [8, 24, 63], combined with process-algebraic reasoning techniques [11, 53, 55, 59, 68]. As far as we know, this is the first time a compositional principle for ensuring SN has been established for name passing processes with non-trivial use of replication. The proof method for SN is applicable to significant extensions of the presented formalism, including state and polymorphism [12, 33, 36, 69, 70]. Further discussions on these extensions are found in Section 7. In the following, we outline key technical ideas and relate our work to the existing literature.

**The π-Calculus.** Following [11], we use an asynchronous variant of the π-calculus [32]. Computation in this calculus is generated by interaction between processes.

$$x(\vec{y}).P \,|\, \overline{x}\langle\vec{v}\rangle \;\longrightarrow\; P\{\vec{v}/\vec{y}\}$$

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 receptor with replication

$$!x(\vec{y}).P \,|\, \overline{x}\langle\vec{v}\rangle \;\longrightarrow\; !x(\vec{y}).P \,|\, P\{\vec{v}/\vec{y}\},$$

where the replicated process remains in the configuration after reduction, which behaves as a shared resources or a remote server. As a simple example of a process, first consider the *forwarder* agent $\mathtt{Fw}\langle ab\rangle$

$$\mathtt{Fw}\langle ab\rangle \stackrel{\text{def}}{=} !a(x).\overline{b}\langle x\rangle$$

which repeatedly inputs a value at $a$ and outputs it immediately at $b$. As another example, the following is a client which requests, via $a$, to have a value returned via a private name

$c$

$$\overline{a}(c)\, c(y).P$$

where $\overline{a}(c)\, c(y).P$ stands for $(\nu c)(\overline{a}\langle c\rangle \mid c(y).P)$ with $(\nu c)$ being a restriction operator. Using these agents, $R$ below is a simple way to represent what may be regarded as a denial of service at $c$.

$$R \stackrel{\text{def}}{=} \mathtt{Fw}\langle aa\rangle \mid \overline{a}(c)\, c(y).P$$

This process does not directly demonstrate circularity as in the example above. However, since $R$ causes an infinite reduction sequence, the receptor $c(y).P$ waits forever for an answer at $c$. In an untyped setting, $R$ is equal to $\overline{a}(c)\, c(y).P$ up to asynchronous bisimilarity [32], but the two are quite different regarding resource consumption. The next example shows how subtleties arise through new link creation of the π-calculus.

$$a(x).\mathtt{Fw}\langle bx\rangle \mid \overline{a}(c)\mathtt{Fw}\langle cb\rangle \mid \overline{b}$$

After one step reduction via $a$, we obtain

$$\mathtt{Fw}\langle bc\rangle \mid \mathtt{Fw}\langle cb\rangle \mid \overline{b}$$

which exhibits divergence.

**Type Discipline for SN.** The type discipline of this paper is a simple refinement of [11]. Concretely, the system is based on two central ideas:

(1) *Linear types* [23, 43, 45, 68], which ensure that a channel is used exactly once for input/output and, for a replicated channel, an input occurs exactly once and output occurs zero or more times [11, 41, 52, 55, 59].

(2) *Action types with causality*, where causality is represented by edges in a directed graph whose acyclicity ensures the absence of circular dependencies [43, 45, 68], cf. [23]. Transmission of causality is controlled by a form of *cut elimination* in action types.

Let us illustrate these points by examples. In the standard typing system of the π-calculus [52, 66], $\mathtt{Fw}\langle ab\rangle$ is typed as follows:

$$\vdash \mathtt{Fw}\langle ab\rangle \triangleright a:(\tau), b:(\tau)$$

where $a:(\tau)$ represents that name $a$ inputs or outputs value with type $\tau$. As first refinement, we attach *action modes* to types to ensure the linearity of channels. To simplify discussions, here we use only two modes, "!" and "?", which represent *unique server* and *client requests to server*, respectively. Let us write $\mathtt{Fw}\langle ab\rangle$ for $!a.\overline{b}$, used in the rest of this section (this simplified form of forwarder, which carries no names, is enough for our present discussion). This process has the following type.

$$\vdash \mathtt{Fw}\langle ab\rangle \triangleright a:()^{!}, b:()^{?} \tag{1}$$

This type means that a unique replicated input (server) exists at $a$, and $b$ is a channel that is used for service requests from a unique replicated server at $b$ [10, 11, 28, 59] (a name of type with mode ? can appear many times in the process). When composing processes, cut elimination occurs between input and output on a shared name with dual types. Here ! and ? are dual to each other, cf. [23], so that cut elimination occurs between $()^!$ and $()^?$, resulting in $()^!$ since the server can always consume a client request. Thus the composition of $\mathtt{Fw}\langle ab\rangle$ and $\mathtt{Fw}\langle bc\rangle$ is typed as:

$$\vdash \mathtt{Fw}\langle ab\rangle \,|\, \mathtt{Fw}\langle bc\rangle \triangleright a:()^!, b:()^!, c:()^?. \tag{2}$$

The ideas similar to the refinement above were already presented in [10, 11, 28, 43, 59]. But none of those typing disciplines ensures termination of processes. In fact, the diverging process in (1) is still typable as follows.

$$\vdash \mathtt{Fw}\langle bc\rangle \,|\, \mathtt{Fw}\langle cb\rangle \triangleright b:()^!, c:()^! \tag{3}$$

In the light of such examples, the second refinement introduces the idea to record causality of behaviour in types. For example, $\mathtt{Fw}\langle ab\rangle$ is now typed as follows:

$$\vdash \mathtt{Fw}\langle ab\rangle \triangleright a:()^! \to b:()^?$$

Here $a:()^! \to b:()^?$ indicates that the process repeatedly inputs at $a$ and then outputs at $b$. Cut elimination now occurs between dual input and output by keeping the causality between channels. For example, the composition of two types:

$$a:()^! \to b:()^? \text{ and } b:()^! \to c:()^? \text{ becomes } (a:()^! \to c:()^?),\ (b:()^! \to c:()^?)$$

hence we can type the process in (2) as:

$$\vdash \mathtt{Fw}\langle ab\rangle \,|\, \mathtt{Fw}\langle bc\rangle \triangleright a:()^! \to c:()^?, b:()^! \to c:()^?$$

Now we can detect a cyclic dependency such as $\mathtt{Fw}\langle bc\rangle \,|\, \mathtt{Fw}\langle cb\rangle$ in (3) by looking at their types $b:()^! \to c:()^?$ and $c:()^! \to b:()^?$ [33, 41, 68] (which, when combined, induces a vicious circle). This simple causality information turns the system with possibly diverging processes [10, 11] into a strongly normalising one. The type discipline based on these ideas is formally presented in Section 2.

**Proving SN for the $\pi$-Calculus.** To prove SN for typable processes, the first idea would be, in the light of the previous examples, to show that reduction steps follow a non-circular ordering on free channels. For example, the reductions of $\overline{a}|\mathtt{Fw}\langle ab\rangle|\mathtt{Fw}\langle bc\rangle$ proceed at $a$, $b$ and $c$ in this order, and eventually terminate. But reductions of $\overline{a}|\mathtt{Fw}\langle ab\rangle|\mathtt{Fw}\langle ba\rangle$ repeat between $a$ and $b$ due to an obvious circularity between $a$ and $b$. However, due to creation of new links and replication of terms, both being crucial features of $\pi$-calculi, considering only simple name ordering is infeasible, at least in its naive form. To see this, consider the following process which only adds one name restriction "$\overline{a}(c)$" to CCS term:

$$!a(x).(\overline{x}|\overline{x}) \,|\, \overline{a}(c)\mathtt{Fw}\langle cb\rangle \,|\, !b.(\overline{a}\langle d\rangle|\overline{a}\langle d\rangle) \tag{4}$$

(this process is typable by $a:((()^?)^!, b:()^! \to d:()^?$ as we shall see later). The process owns reductions first at $a$, then at $b$, then at $a$ again. Further, the number of redexes increases exponentially in its course, *but* the computation terminates. Such behaviour occurs when a process requests the same resource more than once in an interaction, e.g. in an encoding of the $\lambda$-term $\lambda xyz.((xz)(yz))$ [51]. The difficulty in analysing (4) can be seen by considering the following subterm of a one step descendant of (4).

$$(\nu c)(\overline{c} \,|\, \overline{c} \,|\, \mathtt{Fw}\langle cb \rangle)$$

It contains a chain $c \to b$, which is difficult to determine before $c$ is passed. But if we naively represent causality incorporating bound names in (4), there is a circular chain $a \to c \to b \to a$, although this cycle never arises in actual interaction. How can we then prove termination? Simple structural inductions would not be usable for the same reason they do not work in typed $\lambda$-calculi [8, 21].

The idea we use is suggested by SN proofs for typed $\lambda$-calculi, due to, among others, Tait [63]. His method employs a semantic interpretation of each type $[\![\sigma]\!]$ as a collection of strongly normalising $\lambda$-terms, and shows that all typable terms are indeed in these sets. A key step is to prove that $\lambda x : \sigma.M \in [\![\sigma \to \tau]\!]$ for each $M : \tau$ (for which by induction $M \in [\![\tau]\!]$), which means, by definition, $(\lambda x.M)N \in [\![\tau]\!]$ for each $N \in [\![\sigma]\!]$. But all semantic types have the property that $M\{N/x\} \in [\![\tau]\!]$ and $(\lambda x.M)N \longrightarrow M\{N/x\}$ imply $(\lambda x.M)N \in [\![\tau]\!]$, hence we have only to show $M\{N/x\} \in [\![\tau]\!]$. To be able to do this we strengthen the induction hypothesis $M \in [\![\tau]\!]$ to $M \in [\![\tau]\!]_\rho$ for each environment $\rho$, mapping each variable of type $\sigma$ to some term in $[\![\sigma]\!]$. Now the result is immediate [8, 21]. While we cannot use an identical framework due to the different nature of reduction in the $\pi$-calculus, a similar technique works "for the induction to go through". A key observation concerns the close correspondence between the substitution $M\{N/x\}$ and the consumption of a message $\overline{x}\langle v \rangle$ by a replicated process $!x(y).Q$. Thus, at each induction step, we prove that $P|(R_1|...|R_n)$ converges for each possible "environment" $R_1|...|R_n$ which complements $P$. The semantic types of processes are formalised via type-directed predicates which are suggested partly by the original method by Tait [63] and partly by the duality-based method of [1, 23]. Termination behaviour is then calculated via the new reduction relation (called extended reduction), which is suggested by strong bisimilarity and replication theorems [11, 53, 59]. Finally acyclicity in causality yields strong normalisation.

**Summary of Contributions.** The following summaries main technical contributions of the present paper. (4) solves an open problem in [51] for the simple type hierarchy.

(1) Introduction of a $\pi$-calculus with the linear typing in which where strong normalisability is ensured by typability.

(2) Establishment of the proof methodology for strong normalisability of typable processes, combining ideas from traditional SN proofs for typed $\lambda$-calculi with process-theoretic reasoning. We also show the result extends to the linear $\pi$-calculus with free name passing via encoding.

(3) Establishment of the finite axiomatisation of the weak bisimilarity in linear processes as a consequence of strong normalisability. The axiomatisation yields an effective procedure to compute equality over linear processes via their normal forms.

(4) Embedding, using Milner's encoding [51], of the simply typed $\lambda$-calculus with sums and products ($\lambda_{\to,\times,+}$) into our typed $\pi$-calculus. The embedding is fully abstract w.r.t. the observational congruence of $\lambda_{\to,\times,+}$, justifying all commutative conversions and $\eta$-equations [6, 19, 20, 24], automatically leading to SN in the source calculus.

(5) Establishment of a basic interaction-based liveness property in linear processes via their strong normalisability, bridging the traditional notion of SN and one of the basic properties in concurrent, interactive computation.

**Related Work.** Strong normalisation for typed $\lambda$-calculi has been studied extensively in the past; detailed surveys can be found in [8, 21]. The present paper shows that traditional methods for proving SN can be adapted to interacting processes.

Abramsky extends the Curry-Howard correspondence to linear logic [23] using proof expressions (which are proof nets in term form), and proves SN [1], guiding our present usage of acyclicity in names. This programme is taken further with realisability semantics of linear logic in [5] where CCS processes act as realisers, using renaming operators for typed process composition [28]. The operational structure of [5] follows his own $\pi$-calculus encoding of proof nets [2], offering a process-algebraic understanding of semantics of linear logic. The appeal of realisability lies in treating semantics and syntax uniformly on a logical basis. In the context of SN types for the $\pi$-calculus, sharing of names and dynamic link creation in the $\pi$-calculus make it difficult to directly apply the framework in [1, 5], especially when we consider imperative extensions. In comparison, the present work offers a basic type discipline which does not directly correspond to known logical systems but which is based on the standard operators of processes and simple operational principles, resulting in a new effective method to ensure SN for name passing processes, extensible to a broad class of interactive behaviours.

As our initial example of server-client interaction suggests, SN in processes is closely related to liveness. Yoshida [68] presents a typed $\pi$-calculus with a local liveness property. Kobayashi and colleagues [39, 41–43] propose several typing systems which ensure different forms of liveness; for example in [42] time quotas are assigned to communications for this purpose. Sangiorgi [58] proposes a typing system to guarantee what he calls receptiveness, which means that an appropriate input prefix is always available. Unlike the present work, these and other preceding typing systems for $\pi$-calculi [11, 27, 28, 57, 59] neither guarantee SN nor the associated liveness properties for processes involving non-trivial use of replication. As a result, embeddability of, say, $\lambda_{\rightarrow}$ in these systems does not guarantee the SN of the source calculus.

Since the present work was reported in [70], Sangiorgi has proposed an alternative approach towards termination of interactive processes [60]. He explicitly adds a global name ordering [60] to ensure strong normalisation. This ordering is close to a property derived in our typing system, cf. Proposition 2.1. His proofs are similar to ours which use type-directed predicates for termination. Yet his types do not seem to ensure liveness at linear channels. A fully abstract embedding of existing calculi or a finite axiomatisation of weak bisimilarity is not reported in [60]. On the other hand, his system can type processes with first-order state. Our corresponding result is discussed in [70, Section 6]; see also Section 7 and the next paragraph. Although the constructions in [60] assume a global name ordering, the present paper shows that this is not necessary: name orderings can be specified locally, composing processes one by one. This local type checking makes it possible to add richer type structures on top of the core SN-calculus, as discussed below.

A basic feature of our approach is that we construct an integrated calculus combining restricted calculi with clear behavioural articulations in a bottom-up fashion, cf. [10, 36]. For this purpose, this paper starts from investigations of one of the most restrictive behavioural properties, namely confluent, strongly normalising name passing. This approach allows generalisation: starting with this core calculus that exactly encapsulates the notion of types originally found in terminating functions as types for terminating processes, the framework opens a powerful proof methodology for strong normalisability which smoothly extends to other classes of behaviours such as stateful, nondeterministic, polymorphic and concurrent computation. For example, [12] establishes strong normalisability of linear processes with second-order polymorphism by extending the present proof method with reducibility candidates induced by double-negation closure, cf. [23]. Similarly, by adding recent proof techniques for termination in Classical Logic [44, 64], [69] obtains strong-normalisation of first-order state, non-determinism and concurrency. These results can be augmented to proving liveness in the presence of non-termination and non-determinism by mixing type structures [69]. This incremental nature of our type structure leads to significant applications of SN to the semantics of processes. For example, [71] reports a new bisimilarity method associated with the linear type structure and strong normalisability, and presents applications to the semantics of secrecy in programming languages [17, 61, 62]. In another paper [36], we adapt these results in a practical direction, proposing and verifying a new typing systems for secure programming languages based on linear/affine typed π-calculi, where strong normalisability and linearity play a fundamental role in the analysis.

One aspect of our type structure, *input-output modes* (cf. [4, 38, 52, 55]), has an incarnation in the context of Linear Logic, yielding a variant called *Polarised Linear Logic* (LLP) [49, 50], studied by Laurent. Proof nets for LLP are faithfully embeddable in the replicated fragment of the present calculus (i.e. the sub-calculus which only uses !-? types). Acyclicity in name usage in the presented type discipline corresponds to the so-called Lafont-Danos-Regnier condition in proof nets. These connections shed light on the constructions in the present paper from a logical viewpoint. A key difference is that the constructions in LLP bear logical significance, making it an effective medium to relate computation and proofs; whereas the present type discipline captures SN in the framework of basic process-theoretic operators (parallel composition, hiding and prefix). This process-based approach leads to a uniform type discipline integrating SN with other classes of behaviours, including diverging computation, state, non-determinism and concurrency, as explored in [12, 33, 36, 69, 71].

**Structure of the Paper.** This paper is a full version of [70], presenting detailed proofs, further examples and related results. Strong normalisability of the π-calculus with free output, not presented in [70], is discussed in Section 6. One of the main purposes of this paper is to present the central ideas of, and core proof techniques for, the first-order strong normalisable typed π-calculus, as a basis of their further extensions and applications. The reader interested in further work associated with this paper may refer to [12, 30, 33, 36, 69, 71]. In the rest of this paper, Section 2 introduces the syntax and the type discipline of the first-order linear π-calculus. Section 3 proves the main result, strong normalisability. Section 4 presents a complete axiomatisation of weak bisimilarity in linear processes. Section 5 gives a fully abstract encoding of the simply typed λ-calculus with sums and products ($\lambda_{\rightarrow,\times,+}$) in the calculus. Section 6 extends the results in the previous sections to the calculus with free name passing. Section 7 discusses related results, among others establishing the liveness property of linear processes.

## 2. PROCESSES AND TYPING

### 2.1. Syntax and Reduction

Following [10, 33, 70, 71], we use the asynchronous version of the π-calculus [15, 32] with bound output [58]. We can use free output with precisely the same results, as we shall show in Section 6. However, the proofs for the main results (strong normalisability, axiomatisation of bisimilarity and fully-abstract embedding) are more lucidly presented

**(Structural Rules)**

(S0) $P \equiv Q \quad \text{if } P \equiv_\alpha Q$ (S1) $P|\mathbf{0} \equiv P$ (S2) $P|Q \equiv Q|P$

(S3) $P|(Q|R) \equiv (P|Q)|R$ (S4) $(\nu x)\mathbf{0} \equiv \mathbf{0}$ (S5) $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$

(S6) $(\nu x)(P|Q) \equiv ((\nu x)P)|Q \ \ (x \notin \mathsf{fn}(Q))$ (S7) $\overline{x}(\vec{y})\overline{z}(\vec{w})P \equiv \overline{z}(\vec{w})\overline{x}(\vec{y})P \ \ (x,z \notin \{\vec{w}\vec{y}\})$

(S8) $(\nu z)\overline{x}(\vec{y})P \equiv \overline{x}(\vec{y})(\nu z)P \ \ (z \notin \{x\vec{y}\})$ (S9) $\overline{x}(\vec{y})(P|Q) \equiv (\overline{x}(\vec{y})P)|Q \ \ (\{\vec{y}\} \cap \mathsf{fn}(Q) = \emptyset)$

**(Reduction)**

(Com) $x(\vec{y}).P\,|\,\overline{x}(\vec{y})Q \longrightarrow (\nu\,\vec{y})(P|Q)$ (Res) $P \longrightarrow Q \implies (\nu x)P \longrightarrow (\nu x)Q$

(Com$_!$) $!x(\vec{y}).P\,|\,\overline{x}(\vec{y})Q \longrightarrow !x(\vec{y}).P|(\nu\,\vec{y})(P|Q)$ (Out) $P \longrightarrow Q \implies \overline{x}(\vec{y})P \longrightarrow \overline{x}(\vec{y})Q$

(Par) $P \longrightarrow P' \implies P|Q \longrightarrow P'|Q$ (Cong) $P \equiv P' \longrightarrow Q' \equiv Q \implies P \longrightarrow Q$

**FIG. 1.** Reduction and Structural Rules

with bound outputs, cf. [10, 12, 58, 70]. Let $x, y, \ldots, a, b, \ldots$ range over a countable set of names (also called channels). The set of untyped terms, *processes*, is given by the following grammar.

$$
\begin{array}{rcllcllcll}
P & ::= & x(\vec{y}).P & \text{input} & \mid & P\,|\,Q & \text{parallel} & \mid & \mathbf{0} & \text{inaction} \\
 & \mid & \overline{x}(\vec{y})P & \text{output} & \mid & (\nu x)P & \text{hiding} & \mid & !x(\vec{y}).P & \text{replication}
\end{array}
$$

The bound/free names are defined as usual. We assume that names in a vector $\vec{y}$ are pairwise distinct. Up to structural equality, the output $\overline{x}(\vec{y})Q$ acts like $(\nu\,\vec{y})(\overline{x}\langle\vec{y}\rangle|Q)$ in the standard syntax. The reduction relation $\rightarrow$ and the structural congruence $\equiv$ are defined in Figure 1. The multi-step reduction $\twoheadrightarrow$ is given as $\twoheadrightarrow \stackrel{\text{def}}{=} \equiv \cup \longrightarrow^*$. We also write $\longrightarrow^n$ for $n$-steps of $\longrightarrow$.

As a simple example of processes and their reductions, a *copy-cat agent* [38] (called a dynamic link in [13, 47]) denoted by $[a \rightarrow b]$ which links two locations $a$ and $b$. The simplest copy-cat is $a.\overline{b}$. Another example is $[c \rightarrow d] \stackrel{\text{def}}{=} !c(a).\overline{d}(b)b.\overline{a}$ which is a copy-cat from $c$ to $d$. It interacts with messages at $c$ as: $[c \rightarrow d]\,|\,\overline{c}(a)a.\overline{b} \longrightarrow [c \rightarrow d]\,|\,\overline{d}(a)a.\overline{b}$. In § 6, we shall see that a free output "$\overline{u}\langle w\rangle$" is translated into a bound output with copy-cat "$\overline{u}(v)[v \rightarrow w]$". Then *omega agent* is defined as $\Omega_u \stackrel{\text{def}}{=} (\nu y)([u \rightarrow y]|[y \rightarrow u])$, which immediately diverges when it is interrogated at $u$.

## 2.2. Types

**Action Modes.** The following pairs of *action modes* [11, 33] prescribe how each channel is used in typed processes.

| | | | |
|---|---|---|---|
| $\downarrow$ | Linear Input | $\uparrow$ | Linear Output |
| $!$ | Server input | $?$ | Client request to $!$ |

"$\downarrow$" mode is associated with an input (e.g. $x$ in $x(y).P$) and "!" mode is associated with a replicated input (e.g. $x$ in $!x(y).P$). "$\uparrow$" (resp. "?") mode is associated with an output delivered to "$\downarrow$" (resp. "!"). For example, if $\overline{x}(\vec{y})P$ is composed with $x(\vec{y}).P$, then $x$ in $\overline{x}(\vec{y})P$ has a $\uparrow$-mode, and if it is composed with $!x(\vec{y}).P$, then $x$ has a ?-mode. We also use the mode $\updownarrow$ which guarantees uncomposability of linear channels; for example, if $x.\mathbf{0}$ has a $\downarrow$-mode and $\overline{x}$ has a $\uparrow$-mode, then $x.\mathbf{0}\,|\,\overline{x}$ has $\updownarrow$-mode at $x$. The $\updownarrow$-mode at $x$ indicates that the process $x.\mathbf{0}\,|\,\overline{x}$ cannot be composed with any process that has $x$ as a free name.

We let $p,q,\ldots$ range over action modes. If $p \neq \updownarrow$, we write $\overline{p}$ for the *dual* of $p$, a self-inverse map on the action modes such that $\overline{\downarrow} = \uparrow$ and $\overline{!} = ?$. The four modes correspond to $!_1, ?_1, !_\omega$ and $?_\omega$ introduced in [11], except that the present modes indicate true linearity for linear channels (i.e. input and output interact precisely once) rather than affinity (i.e. input and output interact at most once) and lack of divergence for replicated channels.

**Channel Types.** Next we define *channel types* by the following grammar. Below $p_{\mathrm{I}}$ (resp. $p_{\mathrm{O}}$) denotes input (resp. output) modes.

$$\tau \ ::= \ \tau_{\mathrm{I}} \mid \tau_{\mathrm{O}} \mid \updownarrow \qquad \tau_{\mathrm{I}} \ ::= \ (\vec{\tau}_{\mathrm{O}})^{p_{\mathrm{I}}} \qquad \tau_{\mathrm{O}} \ ::= \ (\vec{\tau}_{\mathrm{I}})^{p_{\mathrm{O}}}$$

The IO-alternation constraint (names used for input carry only output names and vice versa) comes from game semantics [4, 35, 38]. This condition is not essential for SN but simplifies presentation and proofs. For characterising sequential interaction, we may add further constraints as in [10]; we do not do so here since the proof structure of strong normalisability does not change by having this constraint. Let $\mathsf{md}(\tau)$ be the outermost mode of $\tau$; for $\updownarrow$ we set $\mathsf{md}(\updownarrow) = \updownarrow$. $\overline{\tau}$ is defined by dualising the modes of types: $\overline{(\tau_1..\tau_n)^p} \stackrel{\text{def}}{=} (\overline{\tau_1}..\overline{\tau_n})^{\overline{p}}$, and $\overline{\updownarrow}$ is undefined. We define $\odot$ as the least commutative partial operation on channel types such that:

$$(1) \quad \tau \odot \overline{\tau} = \updownarrow \quad (\mathsf{md}(\tau) = \downarrow) \qquad\qquad (2) \quad \tau \odot \tau = \tau \text{ and } \tau \odot \overline{\tau} = \overline{\tau} \quad (\mathsf{md}(\tau) = ?)$$

Intuitively, (1) says that once we compose input-output linear channels, the channel becomes uncomposable. (2) says that a server should be unique, but an arbitrary number of clients can request interactions. Note that other compositions are undefined. For example, $!x.\mathbf{0}\,|\,!x.\mathbf{0}$ is never typable because $()^! \odot ()^!$ is undefined, while $\overline{x}\,|\,\overline{x}$ is typable by $x\!:\!()^?$, and $!x.\mathbf{0}\,|\,\overline{x}\,|\,\overline{x}$ by $x\!:\!()^!$. This partial algebra of channel types ensures, among others, determinacy of computation in typable processes by controlling their composability.

**Action Types and their Algebra.** Channel types are assigned to free names of a process to specify possible usage of names. Action types, on the other hand, carry causality information [68] and witness the real usage of channels. We first define action types. An *action type*, denoted $A, B, \ldots$, is a finite directed graph with nodes of the form $x\!:\!\tau$, such that:

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

We write $x \to y$ if $x\!:\!\tau \to y\!:\!\tau'$ for some $\tau$ and $\tau'$. If $x$ occurs in $A$ and for no $y$ we have $y \to x$ then we say $x$ is *active in* $A$. $|A|$ (resp. $\mathsf{fn}(A)$, $\mathsf{active}(A)$, $\mathsf{md}(A)$) denotes the set of nodes (resp. names, active names, modes) in $A$. We often write $x\!:\!\tau \in A$ instead of $x\!:\!\tau \in |A|$, and write $A(x)$ for the channel type assigned to $x$ in $A$. $A/\vec{x}$ is the result of taking off nodes with names in $\vec{x}$ from $A$. $A,B$ is the graph union of $A$ and $B$, with the condition that $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \emptyset$. $x\!:\!\tau \to A$ is a result of adding $x\!:\!\tau$ to $A$ with an edge from $x\!:\!\tau$ to all of $A$'s active nodes. We assume that "$\to$" is stronger than ",": for example, $a\!:\!()^{\downarrow} \to b\!:\!()^{\uparrow}, c\!:\!()^{\uparrow}$ means $(a\!:\!()^{\downarrow} \to b\!:\!()^{\uparrow}), c\!:\!()^{\uparrow}$.

It is sometimes useful to write down action types syntactically, in which case we generate action types from the following grammar:

$$A ::= \emptyset \mid a\!:\!\tau \mid A,B \mid a\!:\!\tau \to (b_1\!:\!\tau_1, b_2\!:\!\tau_2, ..., b_n\!:\!\tau_n)$$

where we assume, in $a\!:\!\tau \to (b_1\!:\!\tau_1, b_2\!:\!\tau_2, ..., b_n\!:\!\tau_n)$, that $\tau$ is of mode $\downarrow$ or ! and, accordingly, $\tau_i$ is of mode $\uparrow$ or ? with $n \le 0$. We allow, in $A,B$, two different names with the same ?-type to occur in both $A$ and $B$; otherwise we prohibit shared usage of names. We shall often use this notation in examples.

The symmetric partial operator $\odot$ on channel types is already given before. We extend this operator to action types as follows. First, a symmetric relation $\asymp$ on action types is defined as follows. $A \asymp B$ iff:

- whenever $x\!:\!\tau \in A$ and $x\!:\!\tau' \in B$, $\tau \odot \tau'$ is defined; and
- whenever $x_1 \to x_2, x_2 \to x_3, \ldots, \to x_n$ alternately in $A$ and $B$ ($n \ge 2$), we have $x_1 \ne x_n$.

Next we extend $\odot$ to action types. $A \odot B$ is defined iff $A \asymp B$ and, if so, is given by the following action type.

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

We can easily check the following property of $\odot$. See Appendix A.2 for the proof.

LEMMA 2.1. *$\odot$ on action types is a symmetric and associative partial operation with identity $\emptyset$.*

We illustrate how this operator works via simple examples.

EXAMPLE 2.1. Figure 2 shows two examples of composition between action types using $\odot$. In the linear case, ordering from/to node $b$ disappears. On the other hand, in the replicated case, we keep the original ordering because $!b(\vec{y}).P$ remains persistent. The same examples can also be written down syntactically, using the grammar of action types introduced before as follows.

$$\begin{array}{rl}
(1) & a\!:\!(\tau)^{\downarrow} \to (b\!:\!(\overline{\tau})^{\uparrow}, c\!:\!(\overline{\tau})^{\uparrow}) \odot b\!:\!(\tau)^{\downarrow} \to (d\!:\!(\overline{\tau})^{\uparrow}, e\!:\!(\overline{\tau})^{\uparrow}) \\
= & a\!:\!(\tau)^{\downarrow} \to (c\!:\!(\overline{\tau})^{\uparrow}, d\!:\!(\overline{\tau})^{\uparrow}, e\!:\!(\overline{\tau})^{\uparrow}),\ b\!:\!\updownarrow \\
(2) & a\!:\!(\tau)^{!} \to (b\!:\!(\overline{\tau})^{?}, c\!:\!(\overline{\tau})^{?}) \odot b\!:\!(\tau)^{!} \to (d\!:\!(\overline{\tau})^{?}, e\!:\!(\overline{\tau})^{?}) \\
= & a\!:\!(\tau)^{!} \to (c\!:\!(\overline{\tau})^{?}, d\!:\!(\overline{\tau})^{?}, e\!:\!(\overline{\tau})^{?}),\ b\!:\!(\tau)^{!} \to (d\!:\!(\overline{\tau})^{?}, e\!:\!(\overline{\tau})^{?})
\end{array}$$

Note shared ?-channels are duplicated in the syntactic representation.

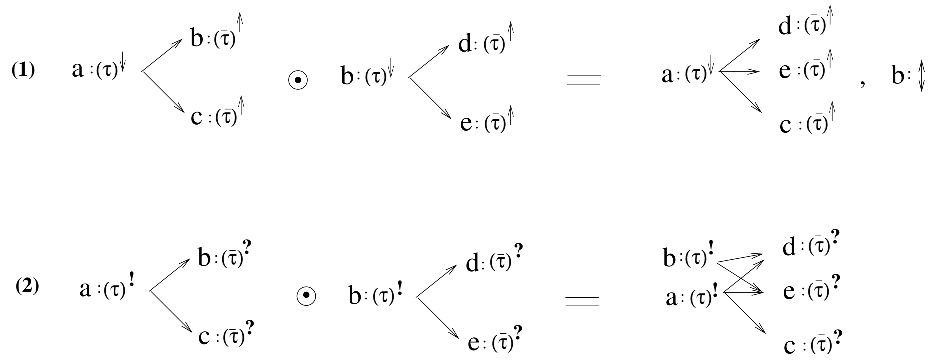


**FIG. 2.** Composition of Action Types

## 2.3. Linear Typing

We are now ready to present the typing rules for strong normalisability. The rules are given in Figure 3, using sequent of the form $\vdash P \triangleright A$.[1] In the typing rules we use the following notations:

- $A^{\vec{y}:\vec{\tau}}$ is $A$ in which each $y_i : \tau_i$ in $\vec{y}:\vec{\tau}$ occurs.
- $A^{-x}$ is $A$ such that $x \notin \mathsf{fn}(A)$.
- $pA$ means $A$ such that $\mathsf{md}(A) = p$.

We say $P$ *is typable under* $A$, or $P$ *has action type* $A$, if $\vdash P \triangleright A$ is derivable. Brief illustration of each rule in Figure 3 follows.

(Zero) starts from the empty action type.

[1]We prefer the format $\vdash P \triangleright A$ to $A \vdash P$. This is because $A$ in $\vdash P \triangleright A$ abstracts the behaviour of $P$ rather than its environment. This point would be elucidated when we discuss translation of $\lambda$-calculus in Section 5.

(Zero)
$$\frac{-}{\vdash \mathbf{0} \triangleright \_}$$

(Par)
$$\frac{\vdash P_i \triangleright A_i \quad (i=1,2) \qquad A_1 \asymp A_2}{\vdash P_1|P_2 \triangleright A_1 \odot A_2}$$

(Res)
$$\frac{\vdash P \triangleright A^{x:\tau} \qquad \mathsf{md}(\tau) \in \{\updownarrow, !\}}{\vdash (\nu x:\tau)P \triangleright A/x}$$

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

(In$^{\downarrow}$)
$$\frac{\vdash P \triangleright \vec{y}:\vec{\tau}, \uparrow A^{-x}, ?B^{-x}}{\vdash x(\vec{y}:\vec{\tau}).P \triangleright (x:(\vec{\tau})^{\downarrow} \to A), B}$$

(In$^{!}$)
$$\frac{\vdash P \triangleright \vec{y}:\vec{\tau}, ?A^{-x}}{\vdash !x(\vec{y}:\vec{\tau}).P \triangleright x:(\vec{\tau})^{!} \to A}$$

(Out)
$$\frac{\vdash P \triangleright A^{\vec{y}:\vec{\tau}} \qquad A \asymp x:(\vec{\tau})^{p\circ}}{\vdash \overline{x}(\vec{y})P \triangleright A/\vec{y} \odot x:(\vec{\tau})^{p\circ}}$$

**FIG. 3.** Linear Typing Rules

(Par) uses $\asymp$ and $\odot$ for controlling composition (which in effect ensures both determinacy and strong normalisability). For example, if $P$ has type $x:()^{\uparrow}$ and $Q$ has type $x:()^{\uparrow}$, then $P\,|\,Q$ is not typable because $()^{\uparrow} \not\asymp ()^{\uparrow}$.

(Res) allows hiding of a name only when its action mode is $\updownarrow$ or ! (which intuitively says that channels of modes $\uparrow$, $\downarrow$ or ? should always be compensated by their duals before they are restricted).

(Weak) weakens $\updownarrow$ and ? since we allow the possibility of having no action at these channels. Formally the weakening of these nodes is necessary for having subject reduction.

(In$^{\downarrow}$) records the causality from linear input type $x:(\vec{\tau})^{\downarrow}$ to linear output types. The side condition $A^{-x}$ and $B^{-x}$ ensure linearity (i.e. unique occurrence) of $x$. For IO-alternation, we let all free names under an input be outputs [10, 11, 36, 71].

(In$^{!}$) records the causality from replicated input type to ?-types. The side condition $A^{-x}$ is required to ensure acyclicity. Of course we cannot allow $\uparrow$-types in the body, for otherwise linearity would be lost. For example, if $z$ is linear channel, then $!x(\vec{y}).(\overline{z}\,|\,Q)$ should be untypable because $z$ is copied at each interaction.

(Out) does not suppress the body by prefix since output is asynchronous. Essentially the rule composes the output prefix and the body in parallel. This rule can be understood by translating $\overline{x}(\vec{y})P$ to $(\nu\,\vec{y})(\overline{x}\langle\vec{y}\rangle\,|\,P)$: suppose $P$ has a type $A$. First we check $A \asymp x:(\vec{\tau})^{p_\circ}$, then if defined, we hide $\vec{y}$ from $A \odot x:(\vec{\tau})^{p_\circ}$, whence $\overline{x}(\vec{y})P$ has type $A/\vec{y} \odot x:(\vec{\tau})^{p_\circ}$.

EXAMPLE 2.2.

- A *copy-cat* copies all information from one channel to another [4, 38], two instances of which already appeared in § 2.1. We show, step by step, how $[u \to x]^{\tau}$ with $\tau = (()^{\uparrow})^{!}$, can be typed:

1: $\vdash \mathbf{0} \triangleright \emptyset$
2: $\vdash \overline{a} \triangleright a:()^{\uparrow}$
3: $\vdash b.\overline{a} \triangleright b:()^{\downarrow} \to a:()^{\uparrow}$
4: $\vdash \overline{x}(b)b.\overline{a} \triangleright x:\overline{\tau}, a:()^{\uparrow}$ (by $(b:()^{\downarrow} \to a:()^{\uparrow})/b = a:()^{\uparrow}$)
5: $\vdash !u(a).\overline{x}(b)b.\overline{a} \triangleright u:\tau \to x:\overline{\tau}$ (by $(x:\overline{\tau}, a:()^{\uparrow})/a = x:\overline{\tau}$)

In this derivation, the length of paths in action types does not exceed 1 even when the term gets bigger and bigger in size. In fact, all paths in action types of derivable sequents have length 0 or 1.

- First we have: $\vdash a.(\overline{b}\,|\,\overline{c}) \triangleright a:()^{\downarrow} \to (b:()^{\uparrow}, c:()^{\uparrow})$ and $\vdash b.\overline{d} \triangleright b:()^{\downarrow} \to d:()^{\uparrow}$. Then

1: $\vdash a.(\overline{b}\,|\,\overline{c})\,|\,b.\overline{d} \triangleright a:()^{\downarrow} \to (c:()^{\uparrow}, d:()^{\uparrow}), b:\updownarrow$ (by (Par))
2: $\vdash (\nu\, b)(a.(\overline{b}\,|\,\overline{c})\,|\,b.\overline{d}) \triangleright a:()^{\downarrow} \to (c:()^{\uparrow}, d:()^{\uparrow})$ (by (Res))

- The connection of two links (copy-cats) is typed as:

$$\vdash [x \to y]^{\tau}\,|\,[y \to z]^{\tau} \triangleright (x:\tau \to z:\overline{\tau}), (y:\tau \to z:\overline{\tau})$$

with $x:\tau \to y:\overline{\tau} \odot y:\tau \to z:\overline{\tau} = x:\tau \to z:\overline{\tau}$. However, $[x \to x]^{\tau}$ and $[x \to y]^{\tau}\,|\,[y \to x]^{\tau}$ which represent cyclic forwarding are untypable by the side condition $A^{-x}$ in (In$^{!}$) and by definition of $\asymp$, respectively.

*Remark.* (type inference) Basically type inference is as difficult as its functional counterpart (i.e. the simply typed $\lambda$-calculus), as already observed in [66]. In our typing system, given modes of all free names in addition to type annotation on bound names in $P$, $\vdash P \triangleright A$ would be linearly decidable. Without information of modes the system has no principal type: for example, $\overline{x}$ could have $()^{\uparrow}$ and $()^{?}$. An interesting topic is to investigate tractable (partial) type inference algorithm for our typing systems.

*Remark.* In [10, 11, 33, 68] as well as in the early version of this paper [70], we used the two-sided sequent $\Gamma \vdash P \triangleright A$ where $\Gamma$ is a standard environment which maps channels to pair types (of the form $\langle\tau, \overline{\tau}\rangle$) and $A$ records the action modes attached to names and causality between names. For example, the copy cat in Example 2.2 (1) is typed as

$$y : \langle\tau,\ \overline{\tau}\rangle, x : \langle\tau,\ \overline{\tau}\rangle \vdash [x \to y]^{\tau} \triangleright !x \to ?y$$

where $\langle\tau,\ \overline{\tau}\rangle$ denotes a pair of input and output types. The typing in this format is similar to those proposed in [18, 39] where types are CCS or $\pi$-terms. Compared to [18, 39], the syntax of action types of the present type discipline (in this format) is simpler since the maximum path length is at most 1; hence an action type is essentially representable as parallel composition of $(!)a.(\overline{b_1} \mid \ldots \mid \overline{b_n})$. The merit of two-sided sequent is its clear division of behavioural constraints into channel types and causality. It may also be useful for type inference. The merit of one-sided sequent is its conciseness and its (potential) faithfulness to semantic content of typed processes. Single- and two-sided sequents result in equivalent typability; the present paper uses the single-sided sequent because it gives a more concise representation of semantic types.

## 2.4. Basic Properties of Typing System

Next we discuss basic properties of the typing system. We begin with name usage in typed processes which form the basis of our later proof of strong normalisability. Below the first property says linear input/output channels and replicated channels occur precisely once in a given process. Acyclicity, the second property, says that the typing rules ensure global partial order between free names via compositional, local type-checking. This property becomes crucial in our SN proof later.

PROPOSITION 2.1. *Let* $\vdash P \triangleright A$.

(1) (linearity) *If* $x : \tau \in A$ *and* $\mathsf{md}(\tau) \in \{\downarrow, \uparrow, !\}$, *$x$ occurs precisely once in $P$.*

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

*Proof.* Both are by induction on the typing rules. (1) is mechanical. For (2), we show that if $\vdash P \triangleright A$ then $x : \tau \to y : \tau'$ in $A$ iff $x \curvearrowright \ldots \curvearrowright y$ is a maximal non-cycle in $P$. This is proven simultaneously with: if there are name-disjoint $x_1 \to x_2,\ x_3 \to x_4,\ \ldots,\ x_{2n-1} \to x_{2n}$

then the corresponding maximal non-cycles do not overlap in names, again by induction on the typing rules. The key case is (Par), the only rule which extends the chain. Assume $x_1 \to x_2, x_2 \to x_3, \ldots x_{n-1} \to x_n$ in $A \uplus B$ and $x_1 \to x_n$ in $A \odot B$. By inductive hypothesis there are the corresponding maximal non-cycles. In them, names used in different non-cycles in $A$ (resp. in $B$) never overlap with each other. Further, since intermediate names in these non-cycles have either mode ! or mode $\updownarrow$, these names do not overlap between $A$ and $B$ either. Thus the result of connecting all these non-cycles again gives a non-cycle, which clearly corresponds to $x_1 \to x_n$, as required. ■

*Remark.* In Proposition 2.1 (2), the notion of chain does not include the case where an intermediate channel is restricted (unlike [1]). While such cases can be included, they are not necessary in the proof of strong normalisability given later, cf. Lemma 3.3. Also note that this property is derived *a posteriori* by defining a composition operator on types, in contrast to [60] which assumes this global condition *a priori*.

Next we list basic properties of the reduction relation in typed processes. In (3) below and henceforth we use the following notations.

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

PROPOSITION 2.2. *Let* $\vdash P \triangleright A$.

(1) (subject reduction) *If* $P \longrightarrow^* Q$ *then* $\vdash Q \triangleright A$.

(2) (strong confluence) *If* $P \longrightarrow Q_i$ $(i = 1, 2)$, *then either* $Q_1 \equiv Q_2$ *or there exists* $R$ *s.t.* $Q_i \longrightarrow R$ $(i = 1, 2)$.

(3) (determinacy) (i) $P \longrightarrow P'$ *and* $\mathsf{SN}(P')$ *imply* $\mathsf{SN}(P)$. (ii) $P \Downarrow Q_i$ $(i = 1, 2)$ *imply* $Q_1 \equiv Q_2$. *And* (iii) $P \Downarrow \Leftrightarrow \mathsf{SN}(P) \Leftrightarrow \mathsf{CSN}(P)$.

*Proof.* (1) uses Lemma 2.1. See Appendix A.2. For (2), we note that the critical pairs arise only when a replicated input is shared which does not change its shape. See Appendix A.3. (3) is standard, cf. [1, 7, 37, 40], all using Proposition 2.2 (2). ■

## 3. STRONG NORMALISATION

This section proves the following result.

THEOREM 3.1. (**main theorem, strong normalisation**) $\vdash P \triangleright A \Rightarrow \mathsf{CSN}(P)$.

A few significant consequences of the theorem will be discussed in Sections 4, 5, 6 and 7. In the proof, we first introduce the *extended reduction relation* $\mapsto$, which eliminates all *cuts* (mutually dual channels) in a typed process. Next we define *semantic types* $[\![A]\!]$, which are sets of typed terms that converge when composed with all necessary "resources" (i.e. complementary processes). Finally we prove that each typable process is in the corresponding semantic type. This part is divided into two stages. We start with showing all normal forms are in their semantic types. Then we establish that each typable process combined with resources always reaches a normal form, which implies strong normalisability of $\longrightarrow$. In the second stage acyclicity of name ordering (cf. Proposition 2.1) becomes crucial: we first define a reduction strategy based on name ordering, then we show any parallel-composed normal forms always reach to a normal form by this strategy.

### 3.1. Extended Reductions

DEFINITION 3.1. (extended reductions) We define $\mapsto_{\mathsf{l}}$, $\mapsto_{\mathsf{r}}$ and $\mapsto_{\mathsf{g}}$ as the typed compatible relations on typed processes modulo $\equiv$ which are generated by the following rules.

$$\begin{array}{llcl}
(\mathsf{E1}) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q & \mapsto_{\mathsf{l}} & C[(\nu\,\vec{y})(P|Q)] \\
(\mathsf{E2}) & C[\overline{x}(\vec{y})P]|!x(\vec{y}).Q & \mapsto_{\mathsf{r}} & C[(\nu\,\vec{y})(P|Q)]\,|\,!x(\vec{y}).Q \\
(\mathsf{E3}) & (\nu\,x)!x(\vec{y}).Q & \mapsto_{\mathsf{g}} & \mathbf{0}
\end{array}$$

Here we assume the term on the left-hand side in each rule is well-typed. $\mapsto \stackrel{\text{def}}{=} (\mapsto_{\mathsf{l}} \cup \mapsto_{\mathsf{r}} \cup \mapsto_{\mathsf{g}})$ is called the *extended reduction* relation. A process is in *extended normal form* if it does not contain $\mapsto$-redex.

The idea of $\mapsto$ is to capture known process-algebraic laws as one step reductions: $\mapsto_{\mathsf{l}}$, $\mapsto_{\mathsf{r}}$ and $\mapsto_{\mathsf{g}}$ correspond to the $\beta$/linear law [27, 28, 43, 68], the replication law [11, 55, 59] and the garbage collection law, respectively. As an example of $\mapsto$, we have:

$$!b(y).\overline{c}(z)z.\overline{y}\,|\,!c(z).\overline{z} \;\mapsto_{\mathsf{r}}\; !b(y).(\nu\,z)(z.\overline{y}\,|\,\overline{z})\,|\,!c(z).\overline{z} \;\mapsto_{\mathsf{l}}\; !b(y).\overline{y}\,|\,!c(z).\overline{z}$$

Immediately $\longrightarrow \subseteq \mapsto$. $P \Downarrow_e$, $\mathsf{SN}_e(P)$ and $\mathsf{CSN}_e(P)$ are given as $P \Downarrow$, $\mathsf{SN}(P)$ and $\mathsf{CSN}(P)$ in §. 2.4 using $\mapsto$ instead of $\longrightarrow$. A $\mapsto$-*redex* is a pair of subterms which form a redex for $\mapsto$ in a given term.

PROPOSITION 3.1. *Let all processes be typed below.*

(1) *If* $\vdash P \triangleright A$ *and* $P \mapsto P'$ *then* $\vdash P' \triangleright A$.
(2) (CR) *If* $P \mapsto^* Q_i$ *then* $Q_i \mapsto^* R$ $(i = 1, 2)$.
(3) (determinacy) *If* $P \mapsto P'$ *and* $\mathsf{SN}_e(P')$ *then* $\mathsf{SN}_e(P)$. *Thus* $P \Downarrow_e$ *iff* $\mathsf{SN}_e(P)$ *iff* $\mathsf{CSN}_e(P)$.
(4) (convergence) (i) $P\,|\,Q \Downarrow_e$ *implies* $P \Downarrow_e$ *and* $Q \Downarrow_e$, (ii) *if* $P \Downarrow_e$, *then* $(\nu x)P \Downarrow_e$, *and* (iii) $P \Downarrow_e$ *iff* $a(\vec{x}).P \Downarrow_e$, (iv) $P \Downarrow_e$ *iff* $!a(\vec{x}).P \Downarrow_e$, *and* (v) $P \Downarrow_e$ *iff* $\overline{a}(\vec{x})P \Downarrow_e$.

*Proof.* See Appendix B.1. The proof of Church-Rosser proceeds by 'postponing' applications of $\mapsto_{\mathsf{g}}$. ■

Note the Church-Rosser property is no longer one-step.

Let us say a process $P$ is *prime with subject* $x$, or simply *prime*, if either $P$ is input with subject $x$ or $P \equiv \overline{x}(y_1..y_n)\Pi_{i\in I}P_i$ such that each $P_i$ is prime with subject $y_i$ where $\Pi_{i\in I}P_i$ denotes the $n$-ary parallel composition of $\{P_i\}_{i\in I}$ (if $I = \emptyset$ then $\Pi_{i\in I}P_i = \mathbf{0}$). In the following proofs we use a variant of the typing rule for output prefixes which is given by adding the condition "$P \equiv \Pi P_i$ with $P_i$ prime with subject $y_i$" in the premise of (Out) in Figure 3. We call this system, *alternative typing system*. Note that, in the alternative typing system, we can assume active names under an output prefix are bound by that prefix. With the same proof as in Appendix D of [11], we can easily check:

PROPOSITION 3.2. *If $\vdash P \triangleright A$ is derivable in the system in Figure 3, then for some $P_0 \equiv P$ we have $\vdash P_0 \triangleright A$ in the alternative typing system.*

Proposition 3.2 says that we can assume, without loss of generality, that all prefixed processes are primes whenever we are discussing properties invariant under $\equiv$ (such as strong normalisability). For this reason the following convention does not lose generality in our technical development.

CONVENTION 3.1. *Hereafter in this section we assume all typed processes are derived in this alternative typing system.*

When we work in the alternative typing system, we restrict $\equiv$ so that it is generated without (S7) and (S9) in Figure 1 (for having closure of typability under $\equiv$ and $\longrightarrow$).

Among others the alternative typing gives a simple inductive characterisation of extended normal forms which we shall use in the proof. Below and henceforth we write $\mathsf{NF}_e$ for the set of extended normal forms: $\mathsf{NF}_e \stackrel{\text{def}}{=} \{P \mid \exists A.\ \vdash P \triangleright A \,\text{and}\, P \not\mapsto\}$.

PROPOSITION 3.3. *$\mathsf{NF}_e$ coincides with, up to $\equiv$, the set of the processes inductively generated by the following rules:*

- $\mathbf{0} \in \mathsf{NF}_e$,
- If $P \in \mathsf{NF}_e$ then $x(\vec{y}:\vec{\tau}).P,\ !x(\vec{y}:\vec{\tau}).P,\ \overline{x}(\vec{y}:\vec{\tau})P \in \mathsf{NF}_e$.
- If $P_i \in \mathsf{NF}_e$ $(i \in I \neq \emptyset)$, $P_i$ is prime, and $P_i|P_j \not\mapsto$ $(i \neq j)$ then $\Pi_{i\in I}P_i \in \mathsf{NF}_e$.

*where we implicitly assume typability in each rule.*

*Proof.* First the set generated is immediately a subset of $\mathsf{NF}_e$ by definition. For the reverse direction, we use induction on tying rules of the (alternative) typing system, noting if $P \in \mathsf{NF}_e$ then its subterms are also in $\mathsf{NF}_e$. For (Res), assume $(\nu x)P$ is derived. Since $P \in \mathsf{NF}_e$, if $x$ has mode $\updownarrow$, this $x$ is the result of weakening, hence the hiding $(\nu x)$ can be taken away by $\equiv$. Further $x$ cannot have mode ! since if so it would result in a $\mapsto$-redex (of rule (E3)). For (Par) assume $P_1|P_2$ is derived. Since each $P_i \in \mathsf{NF}_e$, by induction it is (up to $\equiv$) derived from one of the three rules above. If either is derived from the first one we have nothing to prove. If not, then each is derived from the second or the third rule (up to $\equiv$), and they do not share a complementary channel by $P_1|P_2 \in \mathsf{NF}_e$, thus as required. ■

This proposition says that a process is in $\mathsf{NF}_e$ iff either: (1) it is inaction, (2) it is a prefix of an ENF, or (3) $n$-ary parallel composition of ENFs without complementary input and output. Note this also says that an ENF does not have substantial hiding (i.e. a hiding $(\nu x)P$ such that $x \in \mathsf{fn}(P)$).

### 3.2. Semantic Types

Semantic types are provably strongly normalising typed terms of some kind. We need some preliminaries.

- $\mathsf{cl}(A) \stackrel{\text{def}}{=} \{x_i\!:\!\overline{\tau_i} \mid x_i\!:\!\tau_i \in A,\ \mathsf{md}(\tau_i) \in \{\uparrow, ?\}\}$.
- Let $A \asymp B$ and $A \odot B = C, \vec{x}\!:\!\updownarrow$ where $\updownarrow \notin \mathsf{md}(C)$. Then $A \cdot B \stackrel{\text{def}}{=} C$.

By $\mathsf{cl}(A)$, called the *complement of* $A$, we indicate the (type of the) environment which gives complementary linear and replicated inputs for all free output channels in $A$. $A \cdot B$ is a "semantic version" of $A \odot B$, where we forget inessential $\updownarrow$-channels. Hence by definition, $\mathsf{md}(A \odot B) = !$. We can now define semantic types.

DEFINITION 3.2. Recall $\vec{y}:\vec{\tau} = y_1:\tau_1,...,y_n:\tau_n$. The *semantic type* $[\![A]\!]$ and the *prime semantic type* $\langle\!\langle \vec{x}:\vec{\tau}\rangle\!\rangle$ are defined by the following rules:

$$\begin{aligned}
[\![A]\!] &\stackrel{\text{def}}{=} \{\vdash P \triangleright A \mid \forall Q \in \langle\!\langle \mathsf{cl}(A)\rangle\!\rangle.\ P|Q \Downarrow_e R \in \langle\!\langle A\cdot \mathsf{cl}(A)\rangle\!\rangle\}\\
\langle\!\langle x:(\vec{\tau})^{\downarrow}\rangle\!\rangle &\stackrel{\text{def}}{=} \{x(\vec{y}:\vec{\tau}).P \mid P \in [\![\vec{y}:\vec{\tau}]\!]\}\\
\langle\!\langle x:(\vec{\tau})^{!}\rangle\!\rangle &\stackrel{\text{def}}{=} \{!x(\vec{y}:\vec{\tau}).P \mid P \in [\![\vec{y}:\vec{\tau}]\!]\}\\
\langle\!\langle \vec{x}:\vec{\tau}\rangle\!\rangle &\stackrel{\text{def}}{=} \{\Pi_{i\in I}P_i \mid P_i \in \langle\!\langle x_i:\tau_i\rangle\!\rangle \quad I=\{1,..,n\}\ \}
\end{aligned}$$

Notice that the last definition includes the case $\langle\!\langle\ \rangle\!\rangle = \emptyset$ ($n=0$). Notice that the one-sided sequent offers a simpler form of semantic types than those originally defined in [70].

*Remark.* *(semantic types)* Our definition of semantic types is based on the duality of types and a closure property with respect to a termination predicate $\Downarrow_e$. This construction is partly suggested by Abramsky's term-based reformulation of Girard's method [23], based on double negation closure with respect to a termination predicate [1, § 7.2], and partly by Tait's original method [63]. In their present form, our construction may be more closely related to the latter, due to asymmetry between inputs and outputs (we only have to compose dual input primes each of which has only one active name). For second order polymorphism [12], we use semantic types more closely related to [1, 23]; in the first order setting, the present more compact construction leads to a shorter and simpler proof. A use of the extended reduction relation also leads to a purely rewriting-based proof technique which does not need specific equational semantics as in [1, 60] or auxiliary constructs for representing redexes as in [1, 5].

We can check:

PROPOSITION 3.4. *The rules of $[\![A]\!]$ and $\langle\!\langle \vec{x}:\vec{\tau}\rangle\!\rangle$ are well-defined.*

*Proof.* We formulate a notion of size of types, then verify that each semantic type is defined by semantic types of strictly smaller size. The *size* of $A$ is given by $\sharp(A) \stackrel{\text{def}}{=} \sum_{x\in \mathsf{fn}(A)} A(x)$, where $\sharp((\vec{\tau})^p) \stackrel{\text{def}}{=} \sum_i \sharp(\tau_i)+1$ and $\sharp(\updownarrow)=1$. Write $\leq$ for the ordering w.r.t. the size of types thus defined. By $\sharp(x:(\vec{\tau})^p) \lneq \sum_i \sharp(\vec{y}:\vec{\tau})$, we know that $\langle\!\langle x:(\vec{\tau})^p\rangle\!\rangle$ is defined from types of strictly smaller size, $[\![y_i:\tau_i]\!]$. The same holds for $\langle\!\langle \vec{x}:\vec{\tau}\rangle\!\rangle$. By definition we have $\sharp(A) \geq \sharp(\mathsf{cl}(A)\cdot A) \geq \sharp(\mathsf{cl}(A))$. Thus $[\![A]\!]$ is defined by prime semantic types of the same or smaller size, which, in turn, are defined from those of strictly smaller size. ■

Some observations:

LEMMA 3.1.

(1) *If $P \in [\![A]\!]$ then $\vdash P \triangleright A$ and $\mathsf{SN}_e(P)$.*
(2) *$[\![A]\!] \subseteq [\![A,B]\!]$ and $[\![A]\!] \subseteq [\![B]\!]$ with $A \subseteq B$. Also $[\![A, x:\updownarrow]\!] = [\![A]\!]$.*
(3) *Let $P \mapsto P'$. Then $P \in [\![A]\!]$ iff $P' \in [\![A]\!]$.*
(4) *Let $P_i \in \langle\!\langle x_i:\tau_i\rangle\!\rangle$ $(1 \leq i \leq n)$ such that $x_1,..,x_n$ are pairwise distinct. Then $\Pi_i P_i \in [\![\vec{x}:\vec{\tau}]\!]$.*

*Proof.* For (1), the first half is immediate, while the latter half is because of Proposition 3.1 (3). The first half of (2) is direct from the definition. The latter half is because: $\vdash P \triangleright A, x{:}\updownarrow$ with $x \in \mathsf{fn}(P)$ implies that there always exists a $P'$ s.t. $P \mapsto_{\mathsf{l}} P'$ with $\vdash P' \triangleright A$. For (3), "then" is trivial from the definition of $[\![A]\!]$, while "if" is by CR of $\mapsto$. Finally, for (4), we note $\mathsf{md}(\tau_i) \in \{\downarrow, !\}$, which means $\mathsf{cl}(,x_i{:}\tau_i) = \emptyset$. Hence we can take $Q \equiv \mathbf{0}$ in the definition of $[\![\vec{x}{:}\vec{\tau}]\!]$. ■

### 3.3. Main Proofs

This subsection presents the main arguments for strong normalisability. First we show that all (typable) normal forms are semantically typed. The difficult case here is output $\overline{a}(\vec{x})P$ to replication $!a(\vec{x}).Q$ because after reduction $\overline{a}(\vec{x})P \,|\, !a(\vec{x}).Q \longrightarrow (\nu\vec{x})(P\,|\,Q) \,|\, !a(\vec{x}).Q$, $P$ may interact again with $!a(\vec{x}).Q$. Our formulation of semantic types based on $\mapsto$ makes the inductive argument possible.

First we prove that a typable extended normal form is an element of a semantic type.

LEMMA 3.2. *If* $\vdash P \triangleright A$ *and* $P \in \mathsf{NF}_e$ *then* $P \in [\![A]\!]$.

*Proof.* By Lemma 3.1 (2), it suffices to consider only minimum action types (i.e. $\mathsf{fn}(P) = \mathsf{fn}(A)$). For brevity we write $P_{\langle px \rangle}$ ($p \in \{!, \downarrow\}$) for a process in normal form in a prime semantic type. Also throughout the proof we set $\mathsf{fn}(A) = \{a_i\}$ and $\mathsf{fn}(B) = \{b_j\}$. The proof proceeds by the rule induction on the generation of $\vdash P \triangleright A$ with respect to the rules given in Proposition 3.3. Below given $A = (x_1{:}\tau_1, \ldots, x_n{:}\tau_n)$, we write $\overline{A}$ for $x_1{:}\overline{\tau_1}, \ldots, x_n{:}\overline{\tau_n}$.

**(Inaction)**. By $\mathsf{cl}(\emptyset) = \emptyset$, if $Q \in \langle\!\langle \mathsf{cl}(\emptyset) \rangle\!\rangle$, then $Q \equiv \mathbf{0}$. Hence $\mathbf{0}|Q \equiv \mathbf{0} \Downarrow_e \mathbf{0} \in \langle\!\langle \emptyset \rangle\!\rangle$ with $\mathsf{cl}(\emptyset) \cdot \emptyset = \emptyset$, so that we have $\mathbf{0} \in [\![\emptyset]\!]$.

**(Linear Input)**. Assume $P \in [\![\vec{y}{:}\vec{\tau}, \uparrow A, ?B]\!]$. We show $x(\vec{y}{:}\vec{\tau}).P \in [\![(x{:}(\vec{\tau})^{\downarrow} \to A), B]\!]$. Let $\tau = (\vec{\tau})^{\downarrow}$. First we note $\mathsf{cl}((x{:}\tau \to A), B) = (\overline{A}, \overline{B})$. Let $Q \in \langle\!\langle \overline{A}, \overline{B} \rangle\!\rangle$ and $R \in \langle\!\langle \vec{y}{:}\vec{\overline{\tau}} \rangle\!\rangle$. W.l.o.g. we assume

- $Q \equiv Q_1 | Q_2 \in \mathsf{NF}_e$ such that $Q_1 \equiv \Pi_i Q_{1i\langle \downarrow a_i \rangle} \in \langle\!\langle \overline{A} \rangle\!\rangle$ and $Q_2 \equiv \Pi_j Q_{2j\langle !b_j \rangle} \in \langle\!\langle \overline{B} \rangle\!\rangle$; and
- $R \equiv R_1 \,|\, R_2 \in \mathsf{NF}_e$ such that $R_1 \stackrel{\text{def}}{=} \Pi_k R_{1k\langle \downarrow z_k \rangle}$ with $R_{1k\langle \downarrow z_k \rangle} \in \langle\!\langle z_k{:}\overline{\tau_k} \rangle\!\rangle$ and $R_2 \stackrel{\text{def}}{=} \Pi_l R_{2l\langle !w_l \rangle}$

with $R_{2l\langle !w_l \rangle} \in \langle\!\langle w_l{:}\overline{\tau_l} \rangle\!\rangle$ with $\{\vec{y}\} = \{\vec{z}\vec{w}\}$. Let

By induction hypothesis,

$$P|(Q|R_1 \,|\, R_2) \Downarrow_e Q_2|R_2 \in \langle\!\langle \{w_l{:}\overline{\tau_l}\}_{y_l = w_l}, \overline{B} \rangle\!\rangle \tag{5}$$

Hence by Proposition 3.1 (4-i), we know $P|Q \Downarrow_e P'|Q_2$ where $\mathsf{fn}(P') \subseteq \{\vec{y}\}$. By the definition of $\mapsto$ this implies $x(\vec{y}).P|Q \Downarrow_e x(\vec{y}).P'|Q_2$. We now show $P' \in [\![\vec{y}{:}\vec{\tau}]\!]$, which implies, by definition of $[\![\cdot]\!]$, $x(\vec{y}).P' \in \langle\!\langle x{:}\tau \rangle\!\rangle$. We already know $P|Q|R_1|R_2 \mapsto^* P'|Q_2|R_1|R_2$, while by (5) above, we have $P|Q|R_1|R_2 \mapsto^* P'|Q_2|R_1|R_2 \Downarrow_e Q_2|R_2$. Note that $\mathsf{fn}(Q_2)$ and $\mathsf{fn}(R)$ are disjoint, hence there is no interaction between $Q_2$ and $P'|R_1|R_2$. Now by CR of $\mapsto$ we know $P'|R_1|R_2 \Downarrow_e R_2 \in \langle\!\langle \{w_l{:}\overline{\tau_l}\}_{y_l = w_l} \rangle\!\rangle$. This shows $P' \in [\![\vec{y}{:}\vec{\tau}]\!]$, as required.

**(Replicated Input)**. Similar to the previous case.

**(Linear Output)**. Similar to and simpler than the next case.

**(Replicated Output)**. Assume $P \in [\![C, x{:}(\vec{\tau})^?]\!]$ with $C/\vec{y} = \uparrow A, ?B^{-x}$. Let $\tau = (\vec{\tau})^?$. We have to show $\overline{x}(\vec{y}{:}\vec{\tau})P \in [\![A, B, x{:}\tau]\!]$. First we note that $\mathsf{cl}(A, B, x{:}\tau) = \mathsf{cl}(C, x{:}\tau) = (\overline{A}, \overline{B}, x{:}\overline{\tau})$. Assume $Q \in \langle\!\langle \overline{A}, \overline{B}, x{:}\overline{\tau} \rangle\!\rangle$. W.l.o.g. we can write $Q \equiv !x(\vec{y}).Q_0' \,|\, Q_1 \,|\, Q_2$ where

$!x(\vec{y}).Q_0' \in \langle\langle x:\overline{\tau}\rangle\rangle$, $Q_1 \equiv \Pi_i Q_{1i\langle\downarrow a_i\rangle}$ and $Q_2 \equiv \Pi_j Q_{2j\langle !b_j\rangle}$. Then we have:

$$\overline{x}(\vec{y})P\,|\,Q \;\longrightarrow\; (\nu\,\vec{y})(P|Q_0')\,|\,!x(\vec{y}).Q_0'\,|\,Q_1\,|\,Q_2.$$

By induction hypothesis, $P|!x(\vec{y}).Q_0'|Q_1|Q_2 \Downarrow_e P'|!x(\vec{y}).Q_0'|Q_2$ such that $P' \in [\![\vec{y}:\vec{\tau}]\!]$ with $\mathsf{md}(\tau_i) \in \{!,\downarrow\}$. Hence we can write $P' \equiv \Pi_k R_{1k\langle\downarrow z_k\rangle}\,|\,\Pi_l R_{2l\langle !w_l\rangle}$ with $\{\vec{y}\} = \{\vec{z}\vec{w}\}$. We also note that $Q_0' \in \langle\langle \vec{y}:\vec{\tau}\rangle\rangle$. Hence, by assumption,

$$(\nu\,\vec{y})(P'|Q_0') \mapsto_{\mathsf{l}}^* (\nu\,\vec{y})(\Pi_l R_{2l\langle !w_l\rangle}\,|\,Q_0'') \mapsto^* (\nu\,\vec{y})(\Pi_l R_{2l\langle !w_l\rangle}) \mapsto_{\mathsf{g}}^* \mathbf{0}$$

Now by CR, we have $P\,|\,Q \Downarrow_e !x(\vec{y}).Q_0'\,|\,Q_2 \in \langle\langle \overline{B}, x:\overline{\tau}\rangle\rangle$, as desired.

**(Composition of Primes)**. Given $\Pi_{i\in I}P_i$, assume by induction $P_i \in [\![A_i]\!]$ $(i \in I)$ and $P_i|P_j \not\mapsto$ for $i \neq j$. Note that, for each $x \in \mathsf{fn}(A_i) \cap \mathsf{fn}(A_j)$ $(i \neq j)$ we have $x:\tau \in |A_i| \cap |A_j|$ and $\mathsf{md}(\tau) = ?$. Let $C = A_1 \odot .. \odot A_n$ and $Q \in \langle\langle \mathsf{cl}(C)\rangle\rangle$. W.l.o.g.,

$$Q \;\equiv\; \Pi_{1\le i\le n}(\Pi_{1\le k_i\le m_i} Q'_{k_i\langle\downarrow a_{k_i}\rangle})\,|\,\Pi_j Q''_{j\langle !b_j\rangle}$$

where $\{a_{k_i}\}_{1\le k_i\le m_i}$ is the set of linear output channels in $P_i$ and $\{b_j\}$ is the union of all replicated output channels from $A_1,...,A_n$. By inductive hypothesis (the first part), we have $P_i|Q \Downarrow_e P_i'\,|\,\Pi_{h\neq i}(\Pi_{k_h}Q'_{k_h})$ with $P_i' \in \langle\langle A_i \cdot \mathsf{cl}(A_i)\rangle\rangle$ for each $i$. Since only replicated $Q''_j$ is shared among $P_i$, by $\odot_i \mathsf{cl}(A_i) \supseteq \overline{B}$ and $\odot_i(A_i \cdot \mathsf{cl}(A_i)) = C \cdot \mathsf{cl}(C)$, we have $\Pi P_i\,|\,Q \Downarrow_e \Pi P_i'\,|\,\Pi_j Q''_{j\langle !b_j\rangle} \in \langle\langle C \cdot \mathsf{cl}(C)\rangle\rangle$, as required. We have now exhausted all cases. ■

We use the following corollary of Lemma 3.2.

COROLLARY 3.1.

(1) *Suppose* $\mathsf{fn}(P) = \emptyset$ *and* $P \mapsto^* P' \in [\![\emptyset]\!]$. *Then* $P \Downarrow_e \mathbf{0}$.
(2) *If* $\vdash P \triangleright x:\tau \in \mathsf{NF}_e$ *with* $\mathsf{md}(\tau) \in \{\downarrow, !\}$, *then* $P \in \langle\langle x:\tau\rangle\rangle$.

*Proof.* Both (1) and (2) are straightforward by the above lemma and Lemma 3.1 (3) and (4), respectively. ■

We can now establish the main lemma.

LEMMA 3.3. (main lemma) *Suppose* $\vdash P \triangleright A$. *Then* $P|Q \Downarrow_e$ *for each* $Q \in [\![\mathsf{cl}(A)]\!]$.

Before giving the proof, we discuss its key ideas informally. The proof argues by induction on the typing rules. Given Lemma 3.2, prefix and restriction become trivial, but parallel composition causes a couple of problems. Even if $!a.\overline{b}$ and $(\overline{a}\,|\,!b.\overline{c})$ are in $\mathsf{NF}_e$, their composition (with environment $!c.\mathbf{0}$) allows reductions. How can we prove termination? The key idea is to contract $\mapsto$-redexes from the end of the order of names $c \curvearrowleft b \curvearrowleft a$ as:

$$!a.\overline{b}\,|\,\overline{a}\,|\,!b.\overline{c}\,|\,!c.\mathbf{0} \;\mapsto_{\mathsf{r}}\; !a.\overline{b}\,|\,\overline{a}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0} \;\mapsto_{\mathsf{r}}\; !a.\mathbf{0}\,|\,\overline{a}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0} \;\mapsto_{\mathsf{r}}\; !a.\mathbf{0}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0}$$

and prove that this reduction strategy terminates due to acyclicity of names. This strategy also works for the more complex π-process which corresponds to the term in §.1 (4). Below we underline redexes to be reduced.

$$\begin{array}{rl} !\underline{a}(x).(\overline{x}\,|\,\overline{x})\,|\,\underline{\overline{a}}(c)c.\overline{b}\,|\,!b.\underline{\overline{a}}(d)!d.\overline{y} & \mapsto^2_{\mathsf{r}}\ !a(x).(\overline{x}\,|\,\overline{x})\,|\,(\nu c)(!\underline{c}.\overline{b}\,|\,\underline{\overline{c}}\,|\,\underline{\overline{c}})\,|\,!b.(\nu d)(!\underline{d}.\overline{y}\,|\,\underline{\overline{d}}\,|\,\underline{\overline{d}}) \\ & \mapsto^4_{\mathsf{r}}\ !a(x).(\overline{x}\,|\,\overline{x})\,|\,(\nu \underline{c})(!\underline{c}.\overline{b})\,|\,\overline{b}\,|\,\overline{b}\,|\,!b.((\nu \underline{d})(!\underline{d}.\overline{y})\,|\,\overline{y}\,|\,\overline{y}) \\ & \mapsto^2_{\mathsf{g}}\ !a(x).(\overline{x}\,|\,\overline{x})\,|\,\underline{\overline{b}}\,|\,\underline{\overline{b}}\,|\,!\underline{b}.(\overline{y}\,|\,\overline{y}) \\ & \mapsto^2_{\mathsf{r}}\ !a(x).(\overline{x}\,|\,\overline{x})\,|\,\overline{y}\,|\,\overline{y}\,|\,\overline{y}\,|\,\overline{y}\,|\,!b.(\overline{y}\,|\,\overline{y}) \end{array}$$

The proof follows. Below we say an output channel $x \in \mathsf{fn}(A)$ is *complemented by $R$* if $\vdash R \triangleright \mathsf{cl}(A)$.

*Proof.* By rule induction on the typing rules.

**Case** (Zero). Suppose $\vdash \mathbf{0} \triangleright \emptyset$. Then $\mathsf{cl}(\emptyset) = \emptyset$. Since for all $Q \in [\![\emptyset]\!]$, we have $Q \Downarrow_e \mathbf{0}$ by Corollary 3.1 (1), $\mathbf{0}\,|\,Q \Downarrow_e \mathbf{0}$, as desired.

**Case** (Res). We do case analysis based on the mode of the hidden channel.

**Subcase:** *$\vdash (\nu x)P \triangleright A$ is derived from $\vdash P \triangleright A, x\!:\!\updownarrow$.* We show, for each complementing process $Q \in [\![\mathsf{cl}(A)]\!]$, we have $(\nu x)P|Q \Downarrow_e$. By induction hypothesis, for each $R \in [\![\mathsf{cl}(A, x\!:\!\updownarrow)]\!]$, we have $P|R \Downarrow_e$. Note that $\mathsf{cl}(A, x\!:\!\updownarrow) = \mathsf{cl}(A)$ by definition. Hence, obviously, we have $P|Q \Downarrow_e$ for each $Q \in [\![\mathsf{cl}(A)]\!]$. This in turn implies $(\nu x)P|Q \equiv (\nu x)(P|Q) \Downarrow_e$ by Proposition 3.1 (4-ii), hence done.

**Subcase:** *$\vdash (\nu x)P \triangleright A$ is derived from $\vdash P \triangleright B$ such that* $\mathsf{md}(B(x)) = !$. Without loss of generality, we set $B = A_0 \odot x\!:\!\tau \to ?B_0$ and $A_0 \odot ?B_0 = A$. Again, by definition, we know $\mathsf{cl}(A) = \mathsf{cl}(B)$. The rest is similar to the above case.

**Case** (Weak). Trivial by inductive hypothesis.

**Case** ($\mathsf{In}^{\downarrow}$). Assume $\vdash x(\vec{y}).P \triangleright A$ is derived from $\vdash P \triangleright \vec{y}\!:\!\vec{\tau}, \uparrow A_0^{-x}, ?B_0^{-x}$ with $A = (x\!:\!(\vec{\tau})^{\downarrow} \!\to A_0), B_0$. Let $C = \vec{y}\!:\!\vec{\tau}, A_0, B_0$. By induction hypothesis, for each $Q \in [\![\mathsf{cl}(C)]\!]$, we have $P\,|\,Q \Downarrow_e$, which implies $P \Downarrow_e P' \in \mathsf{NF}_e$ by Proposition 3.1 (4-i). Then by construction of $\mathsf{NF}_e$, we know $a(\vec{y}).P' \in \mathsf{NF}_e$, hence by Lemma 3.2, we know $a(\vec{y}).P' \in [\![A]\!]$. Now by Lemma 3.1 (3), we have $a(\vec{y}).P \in [\![A]\!]$. Then by Lemma 3.1 (1), $a(\vec{y}).P \Downarrow_e$, as desired.

**Case** ($\mathsf{In}^!$). Similar to ($\mathsf{In}^{\downarrow}$).

**Case** (Out). Assume $\vdash \overline{x}(\vec{y})P \triangleright A$ is derived from $\vdash P \triangleright C^{-x}$ such that $\mathsf{active}(C) = \vec{y}$ and $C/\vec{y} = A$. Let $A(x) = \tau$.

**Subcase:** $\mathsf{md}(\tau) = \uparrow$. By induction hypothesis, for each $Q \in [\![\mathsf{cl}(C)]\!]$, we have $P\,|\,Q \Downarrow_e P'\,|\,Q'$ with $P' \in [\![\vec{y} : \vec{\rho}]\!]$ where $\tau = (\vec{\rho})^{\uparrow}$. Assume $R \in [\![\mathsf{cl}(A)]\!]$. Then by the shape of the action type and by definition, we can set $R \stackrel{\text{def}}{=} (x(\vec{y}).R')|Q$ such that $x(\vec{y}).R' \in [\![x : \overline{\tau}]\!]$ and $Q \in [\![\mathsf{cl}(C)]\!]$. We can now calculate:

$$\overline{x}(\vec{y})P|(x(\vec{y}).R')|Q \mapsto^* \overline{x}(\vec{y})P'|(x(\vec{y}).R')|Q' \mapsto (\nu \vec{y})(P'|R')|Q'.$$

By definition $R' \in [\![\vec{y} : \overline{\vec{\rho}}]\!]$, we have $P'|R' \Downarrow_e$. Also by $Q \in [\![\mathsf{cl}(C)]\!]$, we have $Q' \Downarrow_e$. Note that $\mathsf{fn}(Q')$ is disjoint from $\mathsf{fn}(P'|R')$ so that there is no further $\Downarrow_e$ from $(\nu \vec{y})(P'|R')|Q'$. Hence we have $(\nu \vec{y})(P'|R')|Q' \Downarrow_e$, as required.

**Subcase:** $\mathsf{md}(\rho) = ?$. Similar to the subcase above.

**Case** (Par). Suppose $\vdash P_i \triangleright A_i$ with $i = 1, 2$ such that $A_1 \asymp A_2$ and let $A = A_1 \odot A_2$. By induction hypothesis $P_1 \Downarrow_e P_1'$ and $P_2 \Downarrow_e P_2'$. Let $P \stackrel{\text{def}}{=} P_1'|P_2'$. Then $P \equiv Q_1|..|Q_n$ where each $Q_i$ is prime. If $n = 0$ there is nothing to prove. Assume $n \gneq 0$ and let $X \stackrel{\text{def}}{=} \{1, 2, .., n\}$. We define the relation $\searrow$ on $X$ as follows:

$$i \searrow j \ \stackrel{\text{def}}{\Leftrightarrow}\ \exists x \in \mathsf{fn}(Q_i), y \in \mathsf{fn}(Q_j).\ x \curvearrowright y$$

For example, take the process $P \equiv !a.\overline{b}\,|\,\overline{a}\,|\,!b.\overline{c}\,|\,!c.\mathbf{0}$ discussed just before the proof of this lemma, then we have: $1 \searrow 3,\ 2 \searrow 1, 3 \searrow 4$. As in this example, $\searrow^*$ never collapses two names. In fact, if $i \searrow^+ j \searrow^+ i$ then there is a cycle of the form $x \curvearrowright^+ x$ in the sense of Proposition 2.1 (2). Thus the relation $\searrow^*$ is always a partial order on $X$. We now define a series of sets $X_1, X_2, ..$ as follows, writing $\mathsf{max}(Y, \leq)$ for the set of maximal elements of a partially ordered set $Y$.

$$X_1 \stackrel{\text{def}}{=} \mathsf{max}(X, \searrow^*) \qquad X_{i+1} \stackrel{\text{def}}{=} \mathsf{max}(X \setminus \bigcup_{1\leq j\leq i} X_i, \searrow^*)$$

(as a example, $X_1 = 4, X_2 = 3, X_3 = 1, X_4 = 2$ in $P$). As $X$ is finite, $X_1, .., X_m$ partition $X$ for some $m$. Now let $S_i \stackrel{\text{def}}{=} \Pi_{j\in X_i} Q_j$ for $1 \leq i \leq m$. Then $P \equiv \Pi_{1\leq i\leq m} S_i$ and $S_i \in \mathsf{NF}_e$ for each $i$. Choose any $R \in \langle\langle \mathsf{cl}(A)\rangle\rangle$. Note the series $S_1, .., S_n$ is constructed so that outputs in $S_{i+1}$ are always complemented by inputs in $S_i | S_{i-1} | ... | S_1 | R$. Now let $\vdash S_i \triangleright C_i$ s.t. $\odot_{1\leq i\leq m} C_i = A$ and let $E_i \stackrel{\text{def}}{=} \mathsf{cl}(C_1) \odot C_1 \odot .. \odot C_{i-1}$ for $1 \leq i \leq m$. Then $E_i = \mathsf{cl}(C_i)$ for each $i$. Note also $E_1 = \mathsf{cl}(A)$ and $E_m = \mathsf{cl}(A) \odot A$. We now show, by induction on $1 \leq l \leq m+1$, that for some $R_l \in \langle\langle E_l \rangle\rangle$

$$P\,|\,R\ \mapsto^*\ \Pi_{l\leq i\leq m} S_i\,|\,R_l.$$

This proves the lemma when $l = m+1$. For the base case, take $R_1 \equiv R$. For the inductive step, assume $P | R \mapsto^* \Pi_{l\leq i\leq m} S_i | R_l$ such that $R_l \in \langle\langle E_l \rangle\rangle$. By Lemma 3.2 and by $S_l \in \mathsf{NF}_e$ we know that $S_l \in [\![C_l]\!]$. By $E_l = \mathsf{cl}(C_l) = \mathsf{cl}(C_1) \odot C_1 \odot ... \odot C_{l-1}$, this implies $S_l | R_l \Downarrow_e R' \in \langle\langle E_{l+1} \rangle\rangle$. We can now set $R' \equiv R_{l+1}$, as desired. ■

As an immediate corollary, we obtain:

THEOREM 3.2. (strong normalisability in $\mapsto$) $\vdash P \triangleright A$ *implies* $\mathsf{CSN}_e(P)$.

By $\longrightarrow \subseteq \mapsto$ and Proposition 2.2 (3-iii), we have now established Theorem 3.1.

*Remark.* Theorem 3.1 (SN w.r.t. $\longrightarrow$) arises as a corollary of Theorem 3.2 (SN w.r.t. $\mapsto$). This does not mean, however, Theorem 3.1 is of a secondary interest. For example, the liveness property we establish in Section 7 is a direct consequence of Theorem 3.1 rather than that of Theorem 3.2. Further, when state is incorporated, the notion of extended reduction itself becomes inapplicable as it is while Theorem 3.1 and the associated liveness property still make sense. On the other hand, Theorem 3.2 has equational significance, as we shall explore in the next sections.

## 4. CHARACTERISATION OF BISIMILARITY

As a significant consequence of strong normalisability of typed processes, this section shows that weak bisimilarity has a finite axiomatisation in linear processes.

### 4.1. Typed Transitions and Bisimulations

*Typed transitions* describe the observations a typed observer can make of a typed process. The typed transition relation is a proper subset of the untyped transition relation,

$$
\begin{array}{llcll}
(\mathsf{In}) & x(\vec{y}).P^{A} & \stackrel{x(\vec{y})}{\longrightarrow} & P^{\vec{y}:\vec{\tau},A/x} & (x:(\vec{\tau})^{\downarrow}\in A)\\
(\mathsf{Rep}) & !x(\vec{y}).P^{A} & \stackrel{x(\vec{y})}{\longrightarrow} & !x(\vec{y}).P|P^{\vec{y}:\vec{\tau},A} & (x:(\vec{\tau})^{!}\in A)\\
(\mathsf{Out}^{\uparrow}) & \overline{x}(\vec{y})P^{A} & \stackrel{\overline{x}(\vec{y})}{\longrightarrow} & P^{\vec{y}:\vec{\tau},A/x} & (x:(\vec{\tau})^{\uparrow}\in A)\\
(\mathsf{Out}^{?}) & \overline{x}(\vec{y})P^{A} & \stackrel{\overline{x}(\vec{y})}{\longrightarrow} & P^{\vec{y}:\vec{\tau},A} & (x:(\vec{\tau})^{?}\in A)
\end{array}
$$

$$
(\equiv)\,\frac{P_1'\equiv P_1 \quad P_1^{A_1}\stackrel{l}{\longrightarrow}P_2^{A_2}\quad P_2\equiv P_2'}{P_1'^{A_1}\stackrel{l}{\longrightarrow}P_2'^{A_2}}
\qquad
(\mathsf{Res})\,\frac{P_1^{A_1}\stackrel{l}{\longrightarrow}P_2^{A_2}\quad x\notin \mathsf{n}(l)}{(\nu x)P_1^{A_1/x}\stackrel{l}{\longrightarrow}(\nu x)P_2^{A_2/x}}
$$

$$
(\mathsf{Par})\,\frac{P_1^{A_1}\stackrel{l}{\longrightarrow}P_2^{A_2}\quad A_1\odot B \text{ allows } l}{P_1|Q^{A_1\odot B}\stackrel{l}{\longrightarrow}P_2|Q^{A_2\odot B}}
\qquad
(\mathsf{Com})\,\frac{P_1^{A_1}\stackrel{l}{\longrightarrow}P_2^{A_2}\quad Q_1^{B_1}\stackrel{\overline{l}}{\longrightarrow}Q_2^{B_2}}{P_1|Q_1^{A_1\odot B_1}\stackrel{\tau}{\longrightarrow}(\nu\,\mathsf{bn}(l))(P_2|Q_2)^{A_2\odot B_2}}
$$

*A allows l* means (1) if $\mathsf{fn}(l)=\updownarrow$, then $l=\tau$ and (2) if $\mathsf{md}(\mathsf{fn}(l))=!$, then $l$ is not output.

**FIG. 4.** Typed Transition System

while not restricting $\tau$-actions: hence typed transitions restrict observability, not computation. Let the set of *action labels* $l,l',\ldots$ be given by the following grammar.

$$
l \quad ::= \quad \tau \ \mid \ x(\vec{y}) \ \mid \ \overline{x}(\vec{y})
$$

$\mathsf{fn}(l)$ and $\mathsf{bn}(l)$ respectively denote free and bound names in $l$. $\mathsf{n}(l)$ is the set of names in $l$. Using these labels, the typed transition, written $P^A \stackrel{l}{\longrightarrow} Q^B$ where $P^A$ is a shorthand for $\vdash P \triangleright A$, is defined as in Figure 4. Prefix rules are standard, except we do not allow a linear input action and an output action when there is a complementary channel in the process. For example, if a process has $x:\updownarrow$ (resp. $x:(\vec{\tau})^{!}$) in its action type, then both input and output actions (resp. output) at $x$ should be excluded since such actions can never be observed in a typed context (cf. Section 4.2 and Appendix E of [11]). Among the remaining rules, the first rule says that the transition is defined on processes modulo $\equiv$. As we shall discuss later we can dispense with this rule by adding two transition rules for output prefix. The induced transition is well-defined in the following sense.

PROPOSITION 4.1. *If* $\vdash P \triangleright A$ *and* $P^A \stackrel{l}{\longrightarrow} Q^B$ *is derivable from Figure 4 then* $\vdash Q \triangleright B$.

*Proof.* Simple inspection of each rule in Figure 4. ■

In the light of Proposition 4.1, we hereafter safely assume $\vdash P \triangleright A$ and $\vdash Q \triangleright B$ hold whenever we write $P^A \stackrel{l}{\longrightarrow} Q^B$. We also observe:

PROPOSITION 4.2. *Let* $\vdash P \triangleright A$. *Then* $P \longrightarrow Q$ *iff* $P^A \stackrel{\tau}{\longrightarrow} Q^A$.

*Proof.* Standard. In detail, see Appendix C.1. ■

Finally we present the two rules for asynchronous output which allow us to dispense with ($\equiv$) from Figure 4, which becomes useful in our proof later.

$$\frac{P_1^{A_1} \xrightarrow{l} P_2^{A_2} \quad \mathsf{n}(l) \cap \{\vec{y}\} = \emptyset}{\overline{x}(\vec{y})P_1^{A_1/\vec{y}\odot x:(\vec{\tau})^p} \xrightarrow{l} \overline{x}(\vec{y})P_2^{A_2/\vec{y}\odot x:(\vec{\tau})^p}} \qquad \frac{P_1^{A_1} \xrightarrow{x(\vec{z})} P_2^{A_2}}{\overline{x}(\vec{y})P_1^{A_1/\vec{y}\odot x:(\vec{\tau})^p} \xrightarrow{\tau} (\nu\,\vec{y})P_2\{\vec{y}/\vec{z}\}^{A_2/\vec{z}}} \tag{6}$$

These rules materialise asynchronous nature of the output in transition (the second rule needs renaming to avoid clash of bound names). The transition system which adds the rules in (6) to the rules in Figure 4 replacing $\equiv$ in ($\equiv$) by $\equiv_\alpha$, is called *syntactic transition system*. The transition system which simply replaces $\equiv$ in ($\equiv$) by $\equiv_\alpha$ from the rules in Figure 4, is called *prime syntactic transition system*. We observe:

PROPOSITION 4.3.

(1) *If $P^A \xrightarrow{l} Q^B$ in the syntactic transition system, so is in the original system.*

(2) *If $P^A \xrightarrow{l} Q^B$ in the original transition system, then $P^A \xrightarrow{l} Q_0^B$ such that $Q_0 \equiv Q$ in the syntactic transition system.*

(3) *Let $P^A$ be derived under Convention 3.1. Then $P^A \xrightarrow{l} Q^B$ in the original transition system iff $P^A \xrightarrow{l} Q_0^B$ such that $Q_0 \equiv Q$ in the prime syntactic transition system.*

*Proof.* See Appendix C.2. ■

Note (3) indicates that the prime syntactic transition is precisely the transition which corresponds to Convention 3.1.

Based on typed transition, we define a bisimulation. Let us say a relation over typed processes is *typed* if it only relates processes with identical action type. A typed relation is a *typed congruence* when it is a typed equivalence which contains $\equiv$ and which is closed under each typing rule (allowing, as a result, weakening of bases, cf. [11,55]). Below $\stackrel{\hat{l}}{\Longrightarrow}$ denotes the standard abstracted transition.

DEFINITION 4.1. (typed bisimulation) A typed relation $\mathbf{R}$ is a *weak bisimulation*, or a *bisimulation*, if $P_1^{A_1}\mathbf{R}Q_1^{A_1}$ implies: whenever $P_1^{A_1} \xrightarrow{l} P_2^{A_2}$ then there is a typed transition sequence $Q_1^{A_1} \stackrel{\hat{l}}{\Longrightarrow} Q_2^{A_2}$ such that $P_2^{A_2}\mathbf{R}Q_2^{A_2}$, as well as the symmetric case. By replacing $\stackrel{\hat{l}}{\Longrightarrow}$ with $\xrightarrow{l}$, we obtain a *strong bisimulation*. If $P^A\,\mathbf{R}\,Q^A$ for some weak (resp. strong) bisimulation $\mathbf{R}$, we write $P^A \approx Q^A$ (resp. $P^A \sim Q^A$).

We often omit $A$ from $P^A$, writing $P \approx Q$, if $A$ is clear from the context. By definition, $\approx$ (resp. $\sim$) is the union of all weak bisimulations (resp. strong bisimulations), which is in fact the largest weak (resp. strong) bisimulation, and is called *weak* (resp. *strong*) *bisimilarity*. The following technical development focusses on weak bisimilarity, which we hereafter simply call *bisimilarity*. $\approx$ is clearly an equivalence relation. Since $\equiv$ is easily a bisimulation, by Proposition 4.3, it is enough to use the syntactic transition to derive $P \approx Q$ (and the prime one if we are under Convention 3.1).

### 4.2. Axioms

Let $\mathfrak{I}$ ($\mathfrak{I}'$,...) denote a formal (equational) theory over typed processes, which is a set of axioms and rules with formulae of the form $P^A = Q^A$. In $P^A = Q^A$, $P^A$ and $Q^A$ should be well-typed: we shall however not mention types unless they are necessary, writing $P = Q$. If $P = Q$ is provable in $\Im$, we write $\Im \vdash P = Q$. $\Im + \Im'$ is the result of adding the axioms and rules from two theories. We extend this to an arbitrary family of theories.

**Axioms I: (Pre)Congruence Rules.** We consider the standard equivalence rules and closure under well-typed contexts. This theory is denoted $\Im_c$. We also define its subtheory $\Im_p$ by removing (C1) from $\Im_c$.

$$
\begin{array}{llll}
\text{(C1)} & P = Q \;\Rightarrow\; Q = P & \text{(C2)} & P = Q,\ Q = R \;\Rightarrow\; P = R \\
\text{(C3)} & P = Q \;\Rightarrow\; P \,|\, R = Q \,|\, R & \text{(C4)} & P = Q \;\Rightarrow\; R \,|\, P = R \,|\, Q \\
\text{(C5)} & P = Q \;\Rightarrow\; (\nu x)P = (\nu x)Q & \text{(C6)} & P = Q \;\Rightarrow\; x(\vec{y}).P = x(\vec{y}).Q \\
\text{(C7)} & P = Q \;\Rightarrow\; \overline{x}(\vec{y})P = \overline{x}(\vec{y})Q & \text{(C8)} & P = Q \;\Rightarrow\; !x(\vec{y}).P = !x(\vec{y}).Q
\end{array}
$$

**Axioms II: Structural Rules.** Let $\Im_s$ denote the set of rules derived from the axioms (S0–9) in Figure 1. Hence $P \equiv Q$ stands for $\Im_c + \Im_s \vdash P = Q$.

**Axioms III: Conversion Rules.** Convertibility is induced by the extended reduction relation, taking (E1–3) from Definition 3.1 as rules. $\Im_e$ denotes the theory. Note $P \mapsto Q$ iff $\Im_p + \Im_s + \Im_e \vdash P = Q$.

DEFINITION 4.2. The typed congruence $\longleftrightarrow$ is defined by the following logical equivalence: $P \longleftrightarrow Q$ iff $\Im_c + \Im_s + \Im_e \vdash P = Q$.

In other words, $\longleftrightarrow$ is the symmetric and transitive closure of $\mapsto \cup \equiv$.

### 4.3. Characterisation and its Proof

We now show that $\longleftrightarrow$ completely characterises bisimilarity.

THEOREM 4.1. *(characterisation of $\approx$)* $\longleftrightarrow\ =\ \approx$.

We prove Theorem 4.1 by showing two inclusions, (1) $\longleftrightarrow\ \subset\ \approx$ and (2) $\longleftrightarrow\ \supset\ \approx$. We call the first inclusion *soundness* and the second one *completeness*. For soundness, we first show $\approx$ is a typed congruence.

PROPOSITION 4.4. *$\approx$ is a typed congruence.*

*Proof.* See Appendix C.3 ■

Next we show:

PROPOSITION 4.5. *If $\Im_e \vdash P = Q$ then $P \approx Q$.*

*Proof.* See Appendix C.4. ■

Since $\longleftrightarrow$ is the congruent closure of $\Im_e$, by Propositions 4.4 and 4.5 we conclude:

COROLLARY 4.1. $\longleftrightarrow\ \subset\ \approx$.

For the reverse inclusion, we reduce the equality by $\longleftrightarrow$ to those over normal forms.

DEFINITION 4.3. Let us write $P \equiv' Q$ for $\Im_c + \{(\mathsf{S0,S2,S3,S5\text{–}9})\} \vdash P = Q$ and $P \rhd Q$ for $\Im_p + \Im_s \vdash P = Q$ (note $\rhd$ is a *precongruence*). We say $P$ is in $\rhd$-*normal form* if: (1) $P \in \mathsf{NF}_e$ and (2) $P \rhd Q$ implies $P \equiv' Q$.

Note $P \equiv' Q$ means that $P$ and $Q$ are essentially identical without changing the size of terms. For $\triangleright$-normal forms we observe:

LEMMA 4.1.

(1) *A process in* $\mathsf{NF}_e$ *is a* $\triangleright$*-normal form if it does not contain* $\mathbf{0}$ *as its proper subterm.*

(2) *If* $\vdash P \triangleright A$ *then there is a* $\triangleright$*-normal form* $Q$ *such that* $P \mapsto^* Q$.

(3) *The set of* $\triangleright$*-normal forms coincide with those processes generated by the rules in Proposition 3.3.*

(4) *If* $\vdash P \triangleright A$ *and* $P$ *is a* $\triangleright$*-normal form then* $P \equiv' P_\downarrow | P_\uparrow | P_! | P_?$ *where:*

$$\begin{array}{rclrcl} P_\downarrow & = & \Pi_{i \in I_\downarrow} y_i(\vec{z}_i).P_i & P_\uparrow & = & \Pi_{i \in I_\uparrow} \overline{y}_i(\vec{z}_i)P_i \\ P_! & = & \Pi_{i \in I_!} !y_i(\vec{z}_i).P_i & P_? & = & \Pi_{i \in I_?} \overline{y}_i(\vec{z}_i)P_i \end{array}$$

*Here* $I_\downarrow, I_\uparrow, I_!, I_?$ *partition the finite set* $I$ *such that* (i) *for all* $i, j \in I \setminus I_?$*:* $i \neq j$ *implies* $x_i \neq x_j$, (ii) *for all* $i \in I_! \cup I_?$ *and all* $j \in I_\downarrow \cup I_\uparrow$*:* $x_i \neq x_j$ *and* (iii) $P_i$ *is in* $\triangleright$*-normal form for all* $i \in I$*. Furthermore,* $P_\downarrow$*,* $P_\uparrow$*,* $P_!$ *and* $P_?$ *are unique up to* $\equiv'$.

(5) *If* $\vdash P \triangleright A$ *is a* $\triangleright$*-normal form and* $P \xrightarrow{l} Q$ *is a transition, then* $l \neq \tau$.

*Proof.* See Appendix C.5. ■

Let $P$ be a $\triangleright$-normal form. Then $P \equiv' P_\downarrow | P_\uparrow | P_! | P_?$ by Lemma 4.1 (4). The right-hand side of this equation is called *normal form decomposition* of $P$, with $P_\downarrow$, $P_\uparrow$, $P_!$ and $P_?$ being, respectively, its $\downarrow$*-component*, $\uparrow$*-component* $!$*-component* and $?$*-component*.

LEMMA 4.2. *Let* $P^i_\downarrow | P^i_\uparrow | P^i_! | P^i_?$ *be a normal form decomposition* $P^i$ $(i = 1, 2)$.

(1) *Assume that* $P^1_\downarrow = \Pi^m_{j=1} y_i(\vec{z}_i).P^1_j$ *and* $P^2_\downarrow = \Pi^n_{j=1} a_i(\vec{b}_i).P^2_j$*. Then* $P^1_\downarrow \approx P^2_\downarrow$ *iff* $m = n$ *and there is a permutation* $\sigma$ *of* $\{1, \ldots, n\}$ *such that* $y_i(\vec{z}_i).P^1_i \approx a_{\sigma(i)}(\vec{b_{\sigma(i)}}).P^2_{\sigma(i)}$ *for all* $i$*. Similarly for* $P^{1,2}_\uparrow$*,* $P^{1,2}_!$ *and* $P^{1,2}_?$.

(2) $P^1 \approx P^2$ *iff* $P^1_\downarrow \approx P^2_\downarrow$*,* $P^1_\uparrow \approx P^2_\uparrow$*,* $P^1_! \approx P^2_!$ *and* $P^1_? \approx P^2_?$.

*Proof.* For (1), the cases for $P^{1,2}_\downarrow$, $P^{1,2}_\uparrow$ and $P^{1,2}_!$ are immediate by considering traces. For $P^{1,2}_?$, we proceed by contradiction. Assume w.l.o.g. $P^1_? \equiv' \overline{x}(\vec{y})P_1 | \overline{x}(\vec{z})P_2 | P'$ while $P^2_? \equiv' \overline{x}(\vec{a})Q | Q'$ such that neither $P'$ nor $Q'$ contain $x$ as an active name. By Lemma 3.2, all active names in $P_1, P_2$ and $Q$ are in $\{\vec{y}\}$, $\{\vec{z}\}$, and $\{\vec{a}\}$, respectively. Typing then ensures that all these active names are inputs. By Lemma 4.1 (5), no process in $\triangleright$-normal form can have a $\tau$-transition. Hence $P^1_?$ and $P^2_?$ cannot have the same set of traces. (2) follows from (1). ■

We now prove the key lemma for completeness. For (2). we can indeed show $\triangleright$-normal forms are a class of processes where $\approx$, $\sim$, $\equiv'$ and $\equiv$ all coincide.

LEMMA 4.3. *Let* $P$ *and* $Q$ *be* $\triangleright$*-normal forms. Then* $P \approx Q$ *iff* $P \equiv' Q$.

*Proof.* The *size* of $P$, $\mathsf{size}(P)$, is the number of constructors in $P$. $\mathsf{size}(P)$ is invariant under $\equiv'$. By induction on $\mathsf{size}(P) + \mathsf{size}(Q)$ we show $\approx \subset \equiv'$. The base case, $\mathsf{size}(P) + \mathsf{size}(Q) = 2$, is immediate. The inductive step uses Lemma 4.2 (1,2) to reduce the argument to each prime component for which, after stripping off the common prefix, we can always use induction. Since $\equiv'$ is easily a bisimulation we also have $\equiv' \subset \approx$, hence done. ■

$$[Var] \quad E, x{:}T \vdash x : T \qquad [Unit] \quad E \vdash () : \mathtt{unit}$$

$$[Lam] \quad \frac{E, x{:}T \vdash M : T'}{E \vdash \lambda x : T.M : T \Rightarrow T'} \qquad [App] \quad \frac{E \vdash M : T \Rightarrow T' \quad E \vdash N : T}{E \vdash MN : T'}$$

$$[Pair] \quad \frac{E \vdash M_i : T_i \ (i = 1,2)}{E \vdash \langle M_1, M_2 \rangle : T_1 \times T_2} \qquad [Proj] \quad \frac{E \vdash M : T_1 \times T_2}{E \vdash \pi_i(M) : T_i \ (i = 1,2)}$$

$$[Inl] \quad \frac{E \vdash M : T_1}{E \vdash \mathtt{inl}(M) : T_1 + T_2} \qquad [Case] \quad \frac{E \vdash M : T_1 + T_2 \quad E, x_i : T_i \vdash M_i : T'}{E \vdash \mathtt{case}\ M\ \mathtt{of}\ \{\mathtt{in}_i(x_i : T_i).M_i\} : T'}$$

**FIG. 5.** Typing Rules for $\lambda_{\rightarrow,\times,+}$

We can now conclude the proof of Theorem 4.1 by establishing the completeness, $\longleftrightarrow\ \supset\ \approx$, and combining it with Proposition 4.1. Assume $P \approx Q$. By Lemma 4.1 (2) we can find $\triangleright$-normal forms $P_{nf}$ and $Q_{nf}$ of $P$ and $Q$, respectively, such that $P \mapsto^* P_{nf}$ and $Q \mapsto^* Q_{nf}$. By Corollary 4.1, we know $P_{nf} \approx Q_{nf}$. But Lemma 4.3 implies that $\approx$ restricted to $\triangleright$-normal forms is contained in $\longleftrightarrow$, hence $P \mapsto^* P_{nf} \longleftrightarrow Q_{nf}$ and $Q \mapsto^* Q_{nf}$ which means $P \longleftrightarrow Q$, as required.

## 5. FULLY ABSTRACT EMBEDDING OF $\lambda_{\rightarrow,\times,+}$

### 5.1. The Functional Calculus

We use the simply typed $\lambda$-calculus with products and sums (written $\lambda_{\rightarrow,\times,+}$ from now on) as a testbed for the expressiveness of the presented calculus, establishing its fully abstract embeddability in the $\pi$-calculus. We have chosen $\lambda_{\rightarrow,\times,+}$ because of its rich type structures and non-trivial equational theory. For simplicity we omit base types other than unit. We review the syntax of types and terms below, with $i$ ranging over $\{1,2\}$.

$$\begin{array}{rcl} T & ::= & \mathtt{unit} \mid T_1 \rightarrow T_2 \mid T_1 \times T_2 \mid T_1 + T_2 \\ M & ::= & x \mid () \mid \lambda x{:}T.M \mid \langle M, N \rangle \mid \pi_i(M) \mid \mathtt{in}_i(M) \mid \mathtt{case}\ L\ \mathtt{of}\ \{\mathtt{in}_i(x_i : T_i).M_i\}_{i \in \{1,2\}} \end{array}$$

We write $M \equiv_\alpha N$ for $\alpha$-equality on terms. A term is *closed* if no variables occur free. The typing rules are standard, which we list in Figure 5 (cf. [25, 54]). We write $E \vdash M : T$ when a term $M$ is typable with type $T$ under a base $E$. We write $C[\ ]_T : T'$ for a (typed) context of type $T'$ with one hole of type $T$. We often omit type annotations from terms and contexts.

The reduction relation, written $\leadsto$, is the least compatible relation which includes:

$$\begin{array}{rrcl} (\beta) & (\lambda x.M)N & \leadsto & M\{N/x\} \\ (\mathrm{proj}_i) & \pi_i \langle M_1, M_2 \rangle & \leadsto & M_i \\ (\mathrm{case}_i) & \mathtt{case}\ \mathtt{in}_i(L)\ \mathtt{of}\ \{\mathtt{in}_i(x_i).N_i\} & \leadsto & N_i\{L/x_i\} \end{array}$$

Other possible notions of reduction include commuting conversions and $\eta$-rules [23]. We take the minimum meaningful reduction for simplicity, but the main technical results in this section hold for all reasonable variations (this is essentially because normal forms of boolean observables are invariant under these rules). We write $M \Downarrow N$ when $M \leadsto^* N$ and $N \not\leadsto$. A *normal form* is a term which has no further reductions. By easy induction on the

**(Typing Rules)**

$$\frac{\vdash P_i \triangleright \vec{y}_i : \vec{\tau}_i, ?A^{\text{-}x}}{\vdash !x[\&(\vec{y}_i).P_i] \triangleright x : [\&_i \vec{\tau}_i]^! \to A}\ (\text{Bra}^!) \qquad \frac{\vdash P \triangleright A^{\vec{y}:\vec{\tau}_j} \quad A \asymp x : [\oplus_i \vec{\tau}_i]^{p\circ}}{\vdash \overline{x}\mathtt{in}_j(\vec{y})P \triangleright A/\vec{y} \odot x : [\oplus_i \vec{\tau}_i]^{p\circ}}\ (\text{Sel})$$

**(Labelled Transition Rules)**

$$\begin{array}{rcll}
x[\&_i \vec{y}_i.P_i]^A & \xrightarrow{x\mathtt{in}_i(\vec{y}_i)} & P_i^{\vec{y}_i:\vec{\tau}_i,A/x} & (x:[\&\vec{\tau}_i]^{\downarrow} \in A) \\
!x[\&_i \vec{y}_i.P_i]^A & \xrightarrow{x\mathtt{in}_i(\vec{y}_i)} & !x[\&_i \vec{y}_i.P_i] | P_i^{\vec{y}_i:\vec{\tau}_i,A} & (x:[\&\vec{\tau}_i]^! \in A) \\
\overline{x}\mathtt{in}_i(\vec{y})P^A & \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} & P^{\vec{y}:\vec{\tau}_i,A/x} & (x:[\oplus_i \vec{\tau}_i]^{\uparrow} \in A) \\
\overline{x}\mathtt{in}_i(\vec{y})P^A & \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} & P^{\vec{y}:\vec{\tau}_i,A} & (x:[\oplus_i \vec{\tau}_i]^? \in A)
\end{array}$$

**FIG. 6.** Typing and Transition Rules for Branching and Selection

structure of terms, a closed normal form of type $T \to T'$ (resp. $T \times T'$, $T + T'$) has shape $\lambda x.M$ (resp. $\langle M,N \rangle$, $\mathtt{in}_i(M)$).

Equality in $\lambda_{\to,\times,+}$ is not as simple as it may look, due to the existence of sums [23]. To have a semantically meaningful equality, we use observation of "values", cf. [54]. Let $\mathsf{true} \stackrel{\text{def}}{=} \mathtt{in}_1(())$ and $\mathsf{false} \stackrel{\text{def}}{=} \mathtt{in}_2(())$, both of type $\mathbb{B}_\lambda \stackrel{\text{def}}{=} \mathtt{unit} + \mathtt{unit}$. Then $E \vdash M \cong_\lambda N : T$ when, for each context $C[\ ]_T : \mathbb{B}_\lambda$ such that $C[M]$ and $C[N]$ are closed, we have $(C[M] \Downarrow \mathsf{true} \ \Leftrightarrow\ C[N] \Downarrow \mathsf{true})$. The same equality is obtained by taking observability at each sum type, justifying all commuting conversions and $\eta$-rules.

### 5.2. Extension with Branching and Selection

Before encoding $\lambda_{\to,\times,+}$, we extend the typed $\pi$-calculus to its full syntax [11] by incorporating branching and selection. Branching is necessary to represent sums in $\lambda_{\to,\times,+}$ and is also used for defining a reduction-based typed congruence [34].

$$\begin{array}{rcl}
P & ::= & \cdots \mid x[\&_i(\vec{y}_i).P_i] \mid !x[\&_i(\vec{y}_i).P_i] \mid \overline{x}\mathtt{in}_i(\vec{y})P \\
\tau_{\mathrm{I}} & ::= & \cdots \mid [\&_i \vec{\tau}_i]^{\downarrow} \mid\ [\&_i \vec{\tau}_i]^! \qquad\qquad \tau_{\mathrm{O}} \ ::= \ \cdots \mid [\oplus_i \vec{\tau}_i]^{\uparrow} \mid\ [\oplus_i \vec{\tau}_i]^?
\end{array}$$

We often omit the indexing set $I$ (which should be either countable or finite) of $x[\&_{i\in I}(\vec{y}_i).P_i]$. $x[\&_{i\in I}(\vec{y}_i).P_i]$ is called *branching*, while $\overline{x}\mathtt{in}_i(\vec{y}:\vec{\tau})P$ is called *selection*. Similarly for $[\&_i \vec{\tau}_i]^p$ and $[\oplus_i \vec{\tau}_i]^p$. $\equiv$ is defined as in Figure 1. The reduction for branching involves selection of one branch, discarding the remaining ones, as well as name passing.

$$x[\&_i(\vec{y}_i).P_i] \,|\, \overline{x}\mathtt{in}_j(\vec{y}_j)\cdot Q \longrightarrow (\nu\, \vec{y}_j)(P_j \,|\, Q)$$
$$!x[\&_i(\vec{y}_i).P_i] \,|\, \overline{x}\mathtt{in}_j(\vec{y}_j)Q \longrightarrow !x[\&_i(\vec{y}_i).P_i] | (\nu\, \vec{y}_j)(P_j | Q)$$

As an example, a *natural number agent*, $[\![n]\!]_u \stackrel{\text{def}}{=} !u(c)\overline{c}\mathtt{in}_n$, acts as a server which necessarily returns a fixed answer, $n$; see [11, 36, 71] for further examples of reductions.

The typing rules for branching/selection are given in Figure 6. $\mapsto$ is extended as in Definition 3.1. Below in (E4) we assume $n$ holes exhaust all occurrences of (linear) $x$; we extend (E1) in a similar way, reducing $n$-holes simultaneously.

$$
\begin{array}{llll}
(\mathsf{E4}) & C[\overline{x}\mathtt{in}_{j_1}(\vec{y}_{j_1})P]..[\overline{x}\mathtt{in}_{j_n}(\vec{y}_{j_n})P]|x[\&_i(\vec{y}_i).Q_i] & \mapsto_{\mathsf{l}} & C[(\nu\vec{y}_{j_1})(P|Q_{j_1})]..[(\nu\vec{y}_{j_n})(P|Q_{j_n})] \\
(\mathsf{E5}) & C[\overline{x}\mathtt{in}_j(\vec{y}_j)P]|!x[\&_i(\vec{y}_i).Q_i] & \mapsto_{\mathsf{r}} & C[(\nu\vec{y}_j)(P|Q_j)]|!x[\&_i(\vec{y}_i).Q_i] \\
(\mathsf{E6}) & (\nu x)!x[(\vec{y})_i.Q_i] & \mapsto_{\mathsf{g}} & \mathbf{0}
\end{array}
$$

The typed transition is defined by extending the set of labels with $x\mathtt{in}_i(\vec{y})$ and $\overline{x}\mathtt{in}_i(\vec{y})$ and by adding the rules in Figure 6. The weak bisimilarity $\approx$ is then defined by the same clause as in Definition 4.1 in Section 4 using the extended transition relation.

The technical development for the full calculus is identical with that for the unary calculus in the preceding sections, except for the following minor changes:

- In Proposition 2.1: In (1), "precisely once" for a $\uparrow$-channel becomes, under a branching input, "precisely once in each branch". In (2), we extend the relation $\curvearrowright$ for branching inputs and outputs.
- In Proposition 3.3 we add the following clauses to the generation rules of $\mathsf{NF}_e$: if $P_i \in \mathsf{NF}_e$ then $x[(\vec{y}_i).P_i]$ and $!x[(\vec{y})_i.P]$ are in $\mathsf{NF}_e$; if $P \in \mathsf{NF}_e$ then $\overline{x}\mathtt{in}_i(\vec{y})P \in \mathsf{NF}_e$.
- In the definition of semantic types (Definition 3.2), we add:

$$
\begin{array}{lcll}
\langle\langle x : [\&_i \vec{\tau}_i]^{\downarrow} \rangle\rangle & \stackrel{\text{def}}{=} & \{x[\&_i(\vec{y} : \vec{\tau}_i).P_i] & \mid P_i \in [\![\vec{y} : \vec{\tau}_i]\!]\} \\
\langle\langle x : [\&_i \vec{\tau}_i]^{!} \rangle\rangle & \stackrel{\text{def}}{=} & \{!x[\&_i(\vec{y} : \vec{\tau}_i).P_i] & \mid P_i \in [\![\vec{y} : \vec{\tau}_i]\!]\}
\end{array}
$$

With these changes, all arguments and results for the unary calculus carry over to the full syntax. We summarise the main syntactic properties below.

PROPOSITION 5.1.

(1) (reduction) *If* $\vdash P \triangleright A$, *then* (i) $P \longrightarrow Q$ *implies* $\vdash Q \triangleright A$ *and* (ii) $P \longrightarrow Q_{1,2}$ *implies either* $Q_1 \equiv Q_2$ *or* $Q_{1,2} \longrightarrow R$ *for some* $R$, *and* (iii) $\mathsf{CSN}(P)$.

(2) (extended reduction) *If* $\vdash P \triangleright A$, *then* (i) $P \mapsto Q$ *implies* $\vdash Q \triangleright A$ *and* (ii) $P \mapsto Q_{1,2}$ *implies* $Q_{1,2} \mapsto^* R$ *for some* $R$, *and* (iii) $\mathsf{CSN}_e(P)$.

(3) (finite axiomatisation) *Let* $\longleftrightarrow = (\mapsto \cup \equiv)^*$. *Then* $\longleftrightarrow\ = \approx$.

Branching allows us to define contextual equality in the strongly normalising processes, using observables at non-trivial branching types. Formally the *contextual congruence* $\cong$ is the maximum typed congruence over (extended) processes satisfying the following condition. Let $\mathbb{B} = [\oplus_{i=1,2}\ ]^{\uparrow}$ below.

$$\text{If } P_1 \Downarrow_x^i \text{ and } P^{x:\mathbb{B}} \cong Q^{x:\mathbb{B}}, \text{ then } Q \Downarrow_x^i \qquad (i = 1,2)$$

where $P \Downarrow_x^i$ means $P \longrightarrow^* \overline{x}\mathtt{in}_i(\vec{y})P'$ and $P^{x:\mathbb{B}} \cong Q^{x:\mathbb{B}}$ relate $P$ and $Q$ typed under $x : \mathbb{B}$. As in bisimilarity, we sometimes simply write $P \cong Q$ for $P^A \cong Q^A$. We observe:

PROPOSITION 5.2.

(1) (maximal consistency) $\cong$ *is maximally consistent in the sense that the only typed congruence which strictly includes* $\cong$ *is the universal relation.*

(2) (context lemma) (a) *Let* $\vdash P_{1,2} \triangleright A$. *Then* $P_1 \cong P_2$ *if and only if, for each* $\vdash R \triangleright \overline{A}, x{:}\mathbb{B}$, $(\nu\,\mathsf{fn}(A))(P_1|R) \Downarrow^i_x$ *iff* $(\nu\,\mathsf{fn}(A))(P_2|R) \Downarrow^i_x$.
(b) *Let* $R$ *be a replicated process with subject* $x$ *and assume* $\vdash C[P]|R|S \triangleright A$ *for some* $A$ *such that* $x$ *does not occur in* $C[\ \cdot\ ]$. *Then, under the standard bound name convention,* $(\nu x)(C[P]|R|S) \cong C[(\nu x)(P|R)]|S$.
(3) (innocuous actions) *If* $\vdash P \triangleright A$ *and* $\mathsf{md}(A) = ?$ *then* $P^A \cong \mathbf{0}^A$.

*Proof.* See Appendix D.1. ■

$\cong$ and $\approx$ are related in the following way.

PROPOSITION 5.3. $\approx\ \subsetneq\ \cong$.

*Proof.* By Proposition 4.4, $\approx$ is a typed congruence and it respects convergence at $\mathbb{B}$ by definition. Since $\cong$ is the maximum such this shows $\approx \subset \cong$. For strictness, take $\vdash \overline{x} \triangleright x{:}()^?$. By Proposition 5.2 (3) this process is $\cong$-equal to $\mathbf{0}$ but clearly $\overline{x} \not\approx \mathbf{0}$. ■

Finally we list processes of specific form used in the encoding later, called *copycat*. A copy-cat dynamically links two locations, which has an origin in *forwarder* in actors as well as in game semantics.

$$
\begin{array}{rcl}
[x \to x']^{(\vec{\tau})^\downarrow} & \stackrel{\text{def}}{=} & x(\vec{y}).\overline{x'}(\vec{y'})\Pi_i[y'_i \to y_i]^{\overline{\tau_i}} \\
[x \to x']^{(\vec{\tau})^!} & \stackrel{\text{def}}{=} & !x(\vec{y}).\overline{x'}(\vec{y'})\Pi_i[y'_i \to y_i]^{\overline{\tau_i}} \\
[x \to x']^{[\&_i\vec{\tau}_i]^\downarrow} & \stackrel{\text{def}}{=} & x[\&_i(\vec{y}_i).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}] \\
[x \to x']^{[\&_i\vec{\tau}_i]^!} & \stackrel{\text{def}}{=} & !x[\&_i(\vec{y}_i).\overline{x'}\mathtt{in}_i(\vec{y'}_i)\Pi_{ij}[y'_{ij} \to y_{ij}]^{\overline{\tau_{ij}}}]
\end{array}
$$

The following property of copy-cats is used later.

PROPOSITION 5.4.

(1) $\vdash [x \to y]^\tau \triangleright x{:}\tau \to (y{:}\overline{\tau}, ?A)$ *for each input type* $\tau$ *and* $?A$ *with* $x, y \notin \mathsf{fn}(A)$.
(2) $(\nu y)(P|[y \to x]^\tau) \mapsto^* P\{x/y\}$ *assuming typability.*

*Proof.* See Appendix D.2. ■

## 5.3. Sequentiality

One of the basic notions we shall use for the proof of the full abstraction, is *sequentiality*. "Sequential" in this context means that processes have at most one active thread: The combination with the sequential type discipline in [10] can realise this behaviour in linear processes. While the full abstraction result is established in the linear π-calculus without the sequentiality constraint, sequentiality plays a crucial role in several arguments. Below we restrict the linear π-calculus to its sequential subsystem following [11] and study its basic properties used in the subsequent proofs.

The first constraint on the linear typing is on channel types.

DEFINITION 5.1. The set of *sequential channel types* is generated by:

- $(\tau_1..\tau_n)^\downarrow$ is sequential if, for each $1 \le i \le n$, $\tau_i$ is sequential and $\mathsf{md}(\tau_i) = ?$; and
- $(\tau_1..\tau_n)^!$ is sequential if, for each $1 \le i \le n$, $\tau_i$ is sequential and, for each $1 \le i \le n-1$, $\mathsf{md}(\tau_i) = ?$ while $\mathsf{md}(\tau_n) = \uparrow$.

$$\text{(Zero)}\ \frac{-}{\vdash_{\mathtt{I}} \mathbf{0} \triangleright \_} \qquad \text{(Par)}\ \frac{\vdash_{\phi_i} P_i \triangleright A_i \quad (i=1,2) \qquad A_1 \asymp A_2 \quad \phi_1 \asymp \phi_2}{\vdash_{\phi_1 \odot \phi_2} P_1 | P_2 \triangleright A_1 \odot A_2} \qquad \text{(Res)}\ \frac{\vdash_{\phi} P \triangleright A^{x:\tau} \qquad \mathsf{md}(\tau) \in \{\updownarrow, !\}}{\vdash_{\phi} (\nu x:\tau) P \triangleright A/x} \qquad \text{(Weak)}\ \frac{\vdash_{\phi} P \triangleright A^{\text{-}x} \qquad \mathsf{md}(\tau) \in \{\updownarrow, ?\}}{\vdash_{\phi} P \triangleright A, x:\tau}$$

$$\text{(In}^{\downarrow}\text{)}\ \frac{\vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}, \uparrow A^{\text{-}x}, ?B^{\text{-}x}}{\vdash_{\mathtt{I}} x(\vec{y}:\vec{\tau}).P \triangleright (x:(\vec{\tau})^{\downarrow} \to A), B} \qquad \text{(In}^{!}\text{)}\ \frac{\vdash_{\mathtt{O}} P \triangleright \vec{y}:\vec{\tau}, ?A^{\text{-}x}}{\vdash_{\mathtt{I}} !x(\vec{y}:\vec{\tau}).P \triangleright x:(\vec{\tau})^{!} \to A} \qquad \text{(Out)}\ \frac{\vdash_{\mathtt{I}} P \triangleright A^{\vec{y}:\vec{\tau}} \quad A \asymp x:(\vec{\tau})^{p_{\circ}}}{\vdash_{\mathtt{O}} \overline{x}(\vec{y}) P \triangleright A/\vec{y} \odot x:(\vec{\tau})^{p_{\circ}}}$$

**FIG. 7.** Linear Sequential Typing

Dually for output types and similarly for branching/selection types (imposing the same constraint for each summand).

The sequent for sequential typing has the form $\vdash_{\phi} P \triangleright A$ where $\phi \in \{\mathtt{I}, \mathtt{O}\}$ is an *IO-mode*, which ensures $P$ contains at most one active thread. $\phi$ obeys the partial algebra $\mathtt{I} \odot \mathtt{I} = \mathtt{I}$ and $\mathtt{I} \odot \mathtt{O} = \mathtt{O} \odot \mathtt{I} = \mathtt{O}$. When $\phi_1 \odot \phi_2$ is defined (that is, if they are not simultaneously output), then we write $\phi_1 \asymp \phi_2$.

The typing rules are given in Figure 7 (the sequential version of (Bra) and (Sel) follow (In) and (Out)). The use of IO-modes in in (Par) ensures single threading since $\mathtt{O} \odot \mathtt{O}$ is undefined. An output (resp. input) can only prefix a body in input (resp. output) mode, resulting in output (resp. input) mode. $\mathbf{0}$ starts from $\mathtt{I}$. Other rules, (Res,Weak), do not change IO-modes. For this system we observe:

PROPOSITION 5.5.

(1) *If* $\vdash_{\phi} P \triangleright A$ *and* $P \longrightarrow Q$ *then* $\vdash_{\phi} Q \triangleright A$. *Similarly if* $\vdash_{\phi} P \triangleright A$ *and* $P \mapsto Q$ *then* $\vdash_{\phi} Q \triangleright A$.
(2) *If* $\vdash_{\phi} P \triangleright A$ *and* $P \longrightarrow Q_{1,2}$ *then* $Q_1 \equiv Q_2$.
(3) *If* $\vdash_{\phi} P \triangleright A$ *then* $\mathsf{CSN}(P)$ *and* $\mathsf{CSN}_e(P)$.

*Proof.* (1) follows the proof of Proposition 2.2 (presented in Appendix A.2 using A.4), incorporating IO-modes in addition. (2) is because there is at most one active output in a sequential process. (3) is immediate since $\vdash_{\phi} P \triangleright A$ implies $\vdash P \triangleright A$ by definition. ■

*Remark.* Proposition 5.5 (2) indicates the sequential nature of dynamics in sequential linear processes: in spite of this, $\mapsto$ gives a way of computing normal forms of sequential processes by parallel reduction.

A significant property is that linear processes typed under sequential channel types are already sequential from a semantic viewpoint.

DEFINITION 5.2. An action type $A$ is *sequential* if all channel types used in $A$ are sequential and, moreover, it does not contain two linear output channels.

Having at most one linear output in an action type (cf. [11, Appendix F]) makes it possible to have inductive definition of sequentialisation, given next.

PROPOSITION 5.6. (sequentialisation) *Given* $\vdash P \triangleright A$ *such that* $A$ *is sequential,* $P \in \mathsf{NF}_e$ *and* $P$ *does not contain hiding, define* $P^\sharp$ *by the following induction, implicitly assuming typability under sequential* $A$ *in each case.*

- $\mathbf{0}^\sharp \stackrel{\text{def}}{=} \mathbf{0}$ *and* $(P|Q)^\sharp \stackrel{\text{def}}{=} P^\sharp|Q^\sharp$.
- $(x(\vec{y}).P)^\sharp \stackrel{\text{def}}{=} x(\vec{y}).P^\sharp$ *and* $(!x(\vec{y}).P)^\sharp \stackrel{\text{def}}{=} !x(\vec{y}).P^\sharp$*, similarly for branching.*
- $(\overline{x}(\vec{y})P)^\sharp \stackrel{\text{def}}{=} \overline{x}(\vec{y})P^\sharp$ *if* $\uparrow \in \mathsf{md}(A)$, $(\overline{x}(\vec{y})P)^\sharp \stackrel{\text{def}}{=} \mathbf{0}$ *if* $\uparrow \notin \mathsf{md}(A)$*, similarly for selection.*

*Then we have* $\vdash_\phi P^\sharp \triangleright A$ *for some* $\phi$ *and, moreover,* $P \cong P^\sharp$.

*Proof.* See Appendix D.3. ■

Using sequentialisation we establish a refined context lemma. We only present the result for processes of the form needed for our later result.

LEMMA 5.1. (sequential context lemma) *Let* $\vdash P_{1,2} \triangleright x{:}\tau \to ?A$ *(with* $\mathsf{md}(\tau) = !$*) such that* $\tau$ *and* $A$ *are sequential. Then* $P_1 \cong P_2$ *iff for each* $\vdash_0 T \triangleright x{:}\overline{\tau}, u{:}\mathbb{B}$ *and for each* $\vdash_{\mathrm{I}} R \triangleright \overline{A}$ *we have* $(\nu\vec{w})(P_1|R|S_1) \Downarrow^i_u \;\Leftrightarrow\; (\nu\vec{w})(P_2|R|S_2) \Downarrow^i_u$ *where* $\mathsf{fn}(A) = \{\vec{w}\}$.

*Proof.* By Proposition 5.2 (2) and by absorbing/garbage collecting processes using extended reduction, we know $P_1 \cong P_2$ iff for each $\vdash T \triangleright x:\overline{\tau}, u:\mathbb{B}$ and for each $\vdash R \triangleright \overline{A}$. By $\approx \subset \cong$ we know $P \mapsto P'$ implies $P \cong P'$, so that we can take $R$ and $T$ to be in $\mathsf{NF}_e$. Using Proposition 5.6 we can further reduce $R$ and $T$ to be sequential processes. ■

## 5.4. Encoding and Soundness

The encoding of $\lambda_{\to,\times,+}$ is given in Figure 8. The encoding of a $\lambda_{\to,\times,+}$-type $T$, written $T^\circ$, maps $T$ to a replicated type. The encoding of a $\lambda_{\to,\times,+}$-term $E \vdash M : T$, written $[\![E \vdash M : T]\!]$ or $[\![M : T]\!]$ for brevity, adapts Milner's call-by-name encoding [51] to our type structure by adding an indirection at each $\lambda$-abstraction. The encoding of terms follows the encoding of types, and uses type information on variables in a $\lambda$-term. The encoding of a base $E$, written $E^\circ$, maps each $x{:}T$ in $E$ to $x{:}\overline{T^\circ}$, dualising the mode. This can be understood as follows: if we have a $\lambda_{\to,\times,+}$-term $x{:}T \vdash M{:}T'$. The corresponding process interacts with a datum of type $T^\circ$ in the environment, and produces a datum of type $T'^\circ$. Thus at $x$, the process itself should have the type which complements $T^\circ$, that is $\overline{T^\circ}$.

PROPOSITION 5.7. *For each* $T$, $T^\circ$ *is a sequential unary channel type of mode* $!$.

*Proof.* By rule induction of the map $(\ )^\circ$. The base case is $\mathtt{unit}^\circ = (()^\uparrow)^!$, which is immediate. For induction, $(T_1 \Rightarrow T_2)^\circ = (\overline{T_1^\circ}(T_2^\circ)^\uparrow)^!$ is sequential iff $T_{1,2}^\circ$ are sequential and have mode $!$, which is the induction hypothesis. Similarly for $(T_1 \times T_2)^\circ$ and $(T_1 + T_2)^\circ$. ■

PROPOSITION 5.8. (syntactic soundness) *If* $E \vdash M : T$ *then* $\vdash [\![M : T]\!]_u \triangleright u{:}T^\circ \to E^\circ$.

*Proof.* See Appendix D.4. ■

Note also $[\![M : T]\!]_u$ has always the shape $!u(\vec{z}).P$. Further $[\![M : T]\!]_u$ is sequentially typable, though we do not use this property in our subsequent proof. This concludes the verification of static properties of the encoding.

For dynamics, we obtain:

PROPOSITION 5.9. *If* $E \vdash M : T$ *and* $M \rightsquigarrow M'$ *then* $[\![M]\!]_u \mapsto^+ [\![M']\!]_u$.

**(Type)** $\mathtt{unit}^\circ \stackrel{\text{def}}{=} (()^\uparrow)^!$ $\qquad (T_1 \Rightarrow T_2)^\circ \stackrel{\text{def}}{=} (\overline{T_1^\circ}(T_2^\circ)^\uparrow)^!$

$(T_1 \times T_2)^\circ \stackrel{\text{def}}{=} ((T_1^\circ T_2^\circ)^\uparrow)^! \qquad (T_1 + T_2)^\circ \stackrel{\text{def}}{=} ([T_1^\circ \oplus T_2^\circ]^\uparrow)^!$

**(Base)** $\emptyset^\circ \stackrel{\text{def}}{=} \emptyset \qquad (E, x{:}T)^\circ \stackrel{\text{def}}{=} E^\circ, x{:}\overline{T^\circ}$

**(Terms)**

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

$$[\![() : \mathtt{unit}]\!]_u \stackrel{\text{def}}{=} !u(x).\overline{x}$$

$$[\![\lambda x : T'.M : T' \Rightarrow T]\!]_u \stackrel{\text{def}}{=} !u(xz).\overline{z}(m)[\![M : T]\!]_m$$

$$[\![MN : T]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).(\nu\, mn)([\![M : T' \Rightarrow T]\!]_m \mid \mathsf{Arg}\langle m, N, \vec{z}\rangle^{T' \Rightarrow T}) \quad (*)$$

$$[\![\langle M_1, M_2\rangle : T \times T']\!]_u \stackrel{\text{def}}{=} !u(c).\overline{c}(m_1 m_2)([\![M_1 : T]\!]_{m_1} \mid [\![M_2 : T']\!]_{m_2})$$

$$[\![\pi_1\ (M) : T]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).(\nu\, m)([\![M : T \times T']\!]_m \mid \mathsf{Proj}_1\langle m, \vec{z}\rangle^T) \quad (*)$$

$$[\![\mathtt{inl}(M) : T + T']\!]_u \stackrel{\text{def}}{=} !u(c).\overline{c}\mathtt{inl}(m)[\![M : T]\!]_m$$

$$[\![\mathsf{case}\ L\ \mathsf{of}\ \{\mathtt{in}_i(x_i : T_i).M_i\}_{i\in\{1,2\}} : T]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).(\nu\, l)([\![L : T_1 + T_2]\!]_l \mid \mathsf{Sum}\langle l, \vec{z}, \{(x_i)M_i\}\rangle^T) \quad (*)$$

$$\mathsf{Arg}\langle m, N, \vec{z}\rangle^{T_1 \Rightarrow T_2} \stackrel{\text{def}}{=} \overline{m}(nc)([\![N : T_1]\!]_n \mid c(w).\mathsf{Msg}\langle w\vec{z}\rangle^{T_2^\circ})$$

$$\mathsf{Proj}_i\langle m, \vec{z}\rangle^T \stackrel{\text{def}}{=} \overline{m}(e)e(v_1 v_2).\mathsf{Msg}\langle v_i \vec{z}\rangle^{T^\circ}$$

$$\mathsf{Sum}\langle l, \vec{z}, \{(x_i)M_i\}\rangle^T \stackrel{\text{def}}{=} \overline{l}(c)c[\&_{i\in\{1,2\}}(x_i).(\nu\, m)([\![M_i : T]\!]_m \mid \mathsf{Msg}\langle m\vec{z}\rangle^{T^\circ})]$$

$$\mathsf{Msg}\langle x\vec{y}\rangle^{(\vec{\tau})^!} \stackrel{\text{def}}{=} \overline{x}(\vec{y}')\prod[y_i' \to y_i]^{\overline{\tau_i}}$$

$(*)$ $\vec{z} = z_1 z_2$ if $T = T_1 \Rightarrow T_2$, else $\vec{z} = z$.

We omit $\mathtt{inr}(M)$ and $\pi_2(M)$.

**FIG. 8.** Encoding of $\lambda_{\to,\times,+}$

*Proof.* See Appendix D.5. ■

*Remark.* Note there is an exact operational correspondence between $\rightsquigarrow$ and $\mapsto$ : $\rightsquigarrow$ is simulated by $\mapsto$ directly, not up to some semantic equality.

COROLLARY 5.1. *$\lambda_{\to,\times,+}$ is strongly normalising.*

*Proof.* Immediate from Theorem 3.2 and Proposition 5.9. ■

Proposition 5.9 and its corollary offer a faithful computational embedding of $\lambda_{\to,\times,+}$ in the $\pi$-calculus: we now show that this also extends to semantics, starting from soundness. To this end we first analyse the inhabitation property of the linear $\pi$-calculus at $\mathbb{B}_\lambda^\circ$.

PROPOSITION 5.10. *If $\vdash P \triangleright u{:}\mathbb{B}_\lambda^\circ$ and $P \in \mathsf{NF}_e$ then either $P \equiv [\![\mathsf{true} : \mathbb{B}_\lambda]\!]_u$ or $P \equiv [\![\mathsf{false} : \mathbb{B}_\lambda]\!]_u$.*

*Proof.* We use Proposition 3.3, noting $\mathbb{B}_\lambda^\circ \stackrel{\text{def}}{=} ([\oplus_{i=1,2}(()^\uparrow)^!]^\uparrow)^!$. Let $P$ be a $\triangleright$-normal form such that $\vdash P \triangleright u{:}\mathbb{B}_\lambda^\circ$. By Proposition 3.3, $P = !u(z).P_1'$ such that $\vdash P_1' \triangleright z{:}[\oplus_{i=1,2}(()^\uparrow)^!]^\uparrow$.

Again by Proposition 3.3 $P_1' = \overline{z}\mathtt{in}_i(w)P_2'$ with $\vdash P_2' \triangleright w:(()^{\uparrow})^!$. This way we reach $P \equiv\, !u(z).\overline{z}inj_i(w)!w(v).\overline{v}$. ■

LEMMA 5.2. (computational adequacy) *Let* $M : \mathbb{B}_\lambda$ *be closed. Then* $M \Downarrow \mathsf{true}$ *iff* $[\![M]\!]_u \Downarrow_e [\![\mathsf{true}]\!]_u$.

*Proof.* Proposition 5.9 gives the "only if" direction, noting $[\![\mathsf{true}]\!]_u \in \mathsf{NF}_e$. For the "if" direction, we first observe the following property.

**Claim.** If $\vdash M : \mathbb{B}_\lambda$ and $M \Downarrow N$ then either $N \equiv_\alpha \mathsf{true}$ or $N \equiv_\alpha \mathsf{false}$.

As mentioned in Section 5.1, closed $\lambda_{\rightarrow,\times,+}$-normal forms of type $\mathtt{unit}$, $T_1 \Rightarrow T_2$, $T_1 \times T_2$ and $T_1 + T_2$ have the shape, respectively, $()$, $\lambda x.M$, $\langle M_1, M_2\rangle$ and $\mathtt{in}_i M$. This is because: if the closed normal form is $MN$, by induction $M$ should be abstraction hence induces a redex, a contradiction; if $\pi(M)$ is a closed normal form $M$ is again so, thus by induction we know $M$ is a pairing which is impossible; similarly for $\mathtt{case}\ L\ \mathtt{of}\ \{\mathtt{in}_i(x_i).M_i\}$ with respect to $L$. Hence $N$ as above should have form $\mathtt{in}_i(N')$ where $N'$ is of type $\mathtt{unit}$ and is again a closed normal form, that is $N' \stackrel{\mathrm{def}}{=} ()$, as required. Suppose $[\![M]\!] \Downarrow [\![\mathsf{true}]\!]_u$ and $M \Downarrow N$. If $N = \mathsf{true}$ we are done. If not, $N = \mathsf{false}$. By Proposition 5.9, we know $[\![M]\!]_u \Downarrow [\![\mathsf{false}]\!]_u$, which contradicts the CR of $\mapsto$, hence done. ■

By the standard argument we obtain:

COROLLARY 5.2. (soundness) $[\![E \vdash M : T]\!]_u \cong [\![E \vdash N : T]\!]_u$ *implies* $E \vdash M \cong_\lambda N : T$.

## 5.5. Completeness

We now tackle a harder direction, the equational completeness of the encoding. While preceding studies of types for the π-calculus have established the soundness of some λ-calculus embeddings, they are rarely complete due to the fine-grained nature of name-passing [68]. The technical development in this subsection shows, following [10], that the duality-based type discipline gives a precise representation of functional strong normalising computation as name-passing processes, leading to full abstraction. The proof uses *finite canonical forms* (FCFs) [4, 38], which are semantically innocuous extension of $\lambda_{\rightarrow,\times,+}$-terms that can cleanly represent linear sequential processes under the encoded $\lambda_{\rightarrow,\times,+}$-typing. Via FCFs, we know all linear sequential processes of $\lambda_{\rightarrow,\times,+}$-types can be decoded back into $\lambda_{\rightarrow,\times,+}$-terms. By sequential context lemma, this is enough to represent all pertinent process contexts as $\lambda_{\rightarrow,\times,+}$-terms, reaching the completeness. In comparison with [10], we entirely argue via syntactic structure without going through innocent functions (even though the definability argument is closely related to the one based on innocent functions in [10]). The grammar of finite canonical forms [4, 38] (FCFs) follows.

$$\begin{array}{rcl} F & ::= & () \mid x \mid \lambda x.F \mid \langle F_1, F_2\rangle \mid \mathtt{in}_i(F) \mid \mathtt{case}\ x\ \mathtt{of}\ \{\mathtt{in}_i(x_i).F_i\} \\ & \mid & \mathtt{let}\ () = z\ \mathtt{in}\ F \mid \mathtt{let}\ x = zF\ \mathtt{in}\ F' \mid \mathtt{let}\ \langle x, y\rangle = z\ \mathtt{in}\ F \end{array}$$

FCFs use three additional constructs, $\mathtt{let}\ () = N\ \mathtt{in}\ M$ (let-unit), $\mathtt{let}\ x = N_1N_2 : S\ \mathtt{in}\ M$ (let-app) and $\mathtt{let}\ \langle x, y\rangle = N\ \mathtt{in}\ M$ (let-prod). We omit their typing rules [4, 38]. Hereafter we only consider well-typed FCFs.

In the context of the functional calculus, we may consider FCFs in terms of their translation into $\lambda_{\to,\times,+}$-terms, which folds "let" constructs using substitutions. The map, denoted $\mathsf{fold}(\,\cdot\,)$, is given as follows.

$$\begin{array}{rcl} \mathsf{fold}(()) & \stackrel{\text{def}}{=} & () \\ \mathsf{fold}(x) & \stackrel{\text{def}}{=} & x \\ \mathsf{fold}(\lambda x.F) & \stackrel{\text{def}}{=} & \lambda x.\mathsf{fold}(F) \\ \mathsf{fold}(\mathtt{in}_i(F)) & \stackrel{\text{def}}{=} & \mathtt{in}_i(\mathsf{fold}(F)) \end{array} \qquad \begin{array}{rcl} \mathsf{fold}(\langle F_1,\, F_2\rangle) & \stackrel{\text{def}}{=} & \langle \mathsf{fold}(F_1),\, \mathsf{fold}(F_2)\rangle \\ \mathsf{fold}(\mathtt{case}\ x\ \mathtt{of}\ \{\mathtt{in}_i(x_i).F\}) & \stackrel{\text{def}}{=} & \mathtt{case}\ x\ \mathtt{of}\ \{\mathtt{in}_i(x_i).\mathsf{fold}(F)\} \\ \mathsf{fold}(\mathtt{let}\ () = z\ \mathtt{in}\ F) & \stackrel{\text{def}}{=} & \mathsf{fold}(F) \\ \mathsf{fold}(\mathtt{let}\ x = zF\ \mathtt{in}\ F') & \stackrel{\text{def}}{=} & \mathsf{fold}(F')\{z\mathsf{fold}(F)/x\} \\ \mathsf{fold}(\mathtt{let}\ \langle x,y\rangle = z\ \mathtt{in}\ F) & \stackrel{\text{def}}{=} & \mathsf{fold}(F)\{\pi_1(z),\pi_2(z)/x,y\} \end{array}$$

By structural induction, we can check $\mathsf{fold}(F)$ is a $\rightsquigarrow$-normal form for each $F$. By combining this folding with $[\![\ ]\!]_u$, we can now encode FCFs into the $\pi$-calculus.

Another way to map FCFs into the $\pi$-calculus is to directly encode FCFs to ENFs. Below we set, w.l.o.g.: for $\mathtt{let}$s, $[\![F]\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P$; and, for $\mathtt{case}$, $[\![F_i]\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P_i$.

$$\begin{array}{rcl} \langle\!\langle () \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(c).\overline{c} \\ \langle\!\langle x \rangle\!\rangle_u & \stackrel{\text{def}}{=} & [u \to x] \\ \langle\!\langle \lambda x.F \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(xc).\overline{c}(f)\langle\!\langle F \rangle\!\rangle_f \\ \langle\!\langle \mathtt{in}_i(F) \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(c).\overline{c}\mathtt{in}_i(f)\langle\!\langle F \rangle\!\rangle_f \\ \langle\!\langle \langle F_1,F_2\rangle \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(c).\overline{c}(f_1f_2)(\langle\!\langle F \rangle\!\rangle_{f_1} | \langle\!\langle F \rangle\!\rangle_{f_2}) \\ \langle\!\langle \mathtt{case}\ z\ \mathtt{of}\ \{\mathtt{in}_i(x_i).F_i\} \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(\vec{w}).\overline{z}(c)c[\&_i(x_i).P_i] \\ \langle\!\langle \mathtt{let}\ () = z : \mathtt{unit}\ \mathtt{in}\ F \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(\vec{w}).\overline{z}(c)c.P \\ \langle\!\langle \mathtt{let}\ x = zF'\ \mathtt{in}\ F \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(\vec{w}).\overline{z}(f'c)(\langle\!\langle F' \rangle\!\rangle_{f'} | c(x).P) \\ \langle\!\langle \mathtt{let}\ \langle x,y\rangle = z\ \mathtt{in}\ F : T \rangle\!\rangle_u & \stackrel{\text{def}}{=} & !u(\vec{w}).\overline{z}(c)c(xy).P \end{array}$$

$\langle\!\langle F \rangle\!\rangle_u$ and $[\![\mathsf{fold}(F)]\!]_u$ semantically coincide:

LEMMA 5.3. *For each $E \vdash F : T$, we have $[\![\mathsf{fold}(F) : T]\!]_u \cong \langle\!\langle F : T \rangle\!\rangle_u$.*

*Proof.* See Appendix D.6. ∎

A fundamental property of FCFs is that we can decode back processes of $\lambda_{\to,\times,+}$-types onto corresponding FCFs. The decoding is done by first choosing sequential processes (which does not lose generality by Proposition 5.6), then transforming them using certain permutation.

PROPOSITION 5.11.

(1) *For each $T$ and $E$, $u{:}T^\circ \to E^\circ$ is sequential in the sense of Definition 5.2.*
(2) *For each $E \vdash F : T$, we have $\vdash_{\mathtt{I}} \langle\!\langle F : T \rangle\!\rangle_u \triangleright u{:}T^\circ \to E^\circ$.*

*Proof.* (1) is from Proposition 5.8 (1). (2) is easy induction. ∎

LEMMA 5.4. *Assume below processes are in $\mathsf{NF}_e$, are sequentially typed with sequential action types, and obey Convention 3.1 and the standard bound name convention.*

(1) (permutation) $\overline{x}(\vec{r}c)(R \mid c[\&_i(\vec{w}_i).\overline{z}(e)!e(\vec{y}).P_i]) \;\cong\; \overline{z}(e)!e(\vec{y}).\overline{x}(\vec{r}c)(\,R\mid c[\&_i(\vec{w}_i).P_i])$ *if $x$ has ?-mode and $z$ has $\uparrow$-mode.*
(2) ($\eta$-expansion, 1) $!u(\vec{x}z).P \cong !u(\vec{x}z).\overline{z}(\vec{m})P'$ *for some $\vec{m}$ and $P'$ if $z$ is typed as $(\vec{\tau})^\uparrow$.*

(3) ($\eta$-expansion, 2) *We say a sequential $P$ is* $\eta$-expanded *if, for each subterm of $P$ of the form $!u(\vec{x}z).P'$ with $z$ typed with a unary linear type, $P'$ has the shape $\overline{z}(\vec{y})Q$. Then for each sequential $P$, there is an $\eta$-expanded $P^\eta$ such that $P \cong P^\eta$.*

*Proof.* (2) and (3) use (1). See Appendix D.7 for detail. ∎

We can now define the reverse map. Let $\vdash P \triangleright u{:}T^\circ \to E^\circ$ such that $P \in \mathsf{NF}_e$ without hiding or redundant $\mathbf{0}$. By Propositions 5.11 (1) and 5.6, we safely assume $P$ is sequentially typed. Further by Lemma 5.4 (3) let $P$ be $\eta$-expanded.[2] Noting these conditions are satisfied by each subterm of $P$, the map $(\!(P)\!)$ is defined by the following induction. In the last four lines we assume $z \notin \{\vec{y}\}$.

$$\begin{array}{rcl}
(\!(!u(c).\overline{c})\!) & \stackrel{\mathrm{def}}{=} & () \\
(\!(!u(xc).\overline{c}(f)P)\!) & \stackrel{\mathrm{def}}{=} & \lambda x.(\!(P)\!) \\
(\!(!u(c).\overline{c}(f_1f_2)(!f_1(\vec{y_1}).P_1|!f_2(\vec{y_2}).P_1))\!) & \stackrel{\mathrm{def}}{=} & \langle (\!(!f_1(\vec{y_1}).P_1)\!),\ (\!(!f_2(\vec{y_2}).P_2)\!)\rangle \\
(\!(!u(c).\overline{c}\mathtt{in}_i(f)P)\!) & \stackrel{\mathrm{def}}{=} & \mathtt{in}_i((\!(P)\!)) \\
(\!(!u(\vec{y}).\overline{z}(c)c.P)\!) & \stackrel{\mathrm{def}}{=} & \mathtt{let}\ () = z\ \mathtt{in}\ (\!(!u(\vec{y}).P)\!) \\
(\!(!u(\vec{y}).\overline{z}(fc)(P|c(x).P'))\!) & \stackrel{\mathrm{def}}{=} & \mathtt{let}\ x = z(\!(P)\!)\ \mathtt{in}\ (\!(!u(\vec{y}).P')\!) \\
(\!(!u(\vec{y}).\overline{z}(c)c(x_1x_2).P)\!) & \stackrel{\mathrm{def}}{=} & \mathtt{let}\ \langle x_1,x_2\rangle = y\ \mathtt{in}\ (\!(!u(\vec{y}).P)\!) \\
(\!(!u(\vec{y}).\overline{z}(c)c[\&_i(x_i).P_i])\!) & \stackrel{\mathrm{def}}{=} & \mathtt{case}\ z\ \mathtt{of}\ \{\mathtt{in}_i(x_i).(\!(!u(\vec{y}).P_i)\!)\}
\end{array}$$

By inspecting each rule, we immediately observe:

PROPOSITION 5.12. *Let $\vdash_{\mathrm{I}} P \triangleright E^\circ \cdot u : T^\circ$. $P \in \mathsf{NF}_e$ and $P$ is $\eta$-expanded. Then* (1) $E \vdash (\!(P)\!) : T$ *and* (2) $P \equiv \langle\!\langle (\!(P)\!) : T \rangle\!\rangle_u$.

A key property for the completeness follows.

COROLLARY 5.3. (definability) *Let $\vdash P \triangleright E^\circ \cdot u : T^\circ$. Then $P \cong [\![M : T]\!]_u$ for some $E \vdash M : T$.*

*Proof.* By Propositions 5.11 (1) and 5.6, take the sequential version of $P$, $P^\sharp$. By Lemma 5.4 (3), further take its $\eta$-expansion, $P^{\sharp\eta}$. Let $M \stackrel{\mathrm{def}}{=} \mathsf{fold}((\!(P^{\sharp\eta})\!))$. Then we have:

$$P \ \cong\ P^{\sharp\eta} \ \equiv\ \langle\!\langle (\!(P^{\sharp\eta})\!) : T \rangle\!\rangle_u \ \cong\ [\![\mathsf{fold}((\!(P^{\sharp\eta})\!)) : T]\!]_u \ \stackrel{\mathrm{def}}{=}\ [\![M : T]\!]_u,$$

as required (the first equation is by Propositions 5.6 and 5.11 (1); the next $\equiv$ is by Lemma 5.4 (3); and the third equation is by Lemma 5.3). ∎

We can now establish the full abstraction. We use the following isomorphism between $\mathbb{B}$ and $\mathbb{B}^\circ_\lambda$ (actually we only need one direction of the isomorphism).

PROPOSITION 5.13. *(isomorphism) Write $C[\ \cdot\ ]^B_A$ for a context whose hole has type $A$ and whose result has type $B$. Define:*

$$\begin{array}{rcl}
C_{\mathsf{f}}[\ \cdot\ ]^{y:\mathbb{B}^\circ_\lambda}_{x:\mathbb{B}} & \stackrel{\mathrm{def}}{=} & !y(c).(\nu x)(x[\&_{i=1,2}.\overline{c}\mathtt{in}_i(f)!f(g).\overline{g}] \mid [\ \cdot\ ]) \\
C_{\mathsf{b}}[\ \cdot\ ]^{x:\mathbb{B}}_{y:\mathbb{B}^\circ_\lambda} & \stackrel{\mathrm{def}}{=} & (\nu y)(\overline{y}(c)c[\&_{i=1,2}.\overline{x}\mathtt{in}_i] \mid [\ \cdot\ ])
\end{array}$$

[2] In fact, we only need $\eta$-expansion for function types, i.e. when a subterm has the form $!u(xz).P'$. Alternatively, we can dispense with $\eta$-expansion by adding an additional syntax to FCFs.

*Then we have:*

(1) $C_{\mathsf{f},\mathsf{b}}$ *are well-typed.*
(2) $P^{x:\mathbb{B}} \Downarrow^1_x$ *iff* $C_{\mathsf{f}}[P]^y_x \Downarrow_e [\![\mathsf{true}]\!]_y$, *dually* $P^{y:\mathbb{B}^\circ_\lambda} \Downarrow_e [\![\mathsf{true}]\!]_y$ *iff* $C_{\mathsf{b}}[P]^x_y \Downarrow^1_x$.
(3) $C_{\mathsf{b}}[C_{\mathsf{f}}[P]^y_x]^x_y \mapsto^+ P$ *for each* $\vdash P \triangleright x:\mathbb{B}$ *and, dually,* $C_{\mathsf{f}}[C_{\mathsf{b}}[Q]^x_y]^y_x \mapsto^+ Q$.

*Proof.* (1) is immediate. For (2) let $\vdash P \triangleright x:\mathbb{B}$. Without loss generality let $P \in \mathsf{NF}_e$, so that $P \equiv \overline{x}\mathtt{in}_i$.

$$\begin{array}{rcl} C_{\mathsf{f}}[P]^y_x & \stackrel{\text{def}}{=} & !y(c).(\nu x)(x[\&_{i=1,2}.\overline{c}\mathtt{in}_i(f)!f(g).\overline{g}] \mid \overline{x}\mathtt{in}_i) \\ & \mapsto^1 & !y(c).\overline{c}\mathtt{in}_i(f)!f(g).\overline{g} \end{array}$$

Similarly $C_{\mathsf{b}}[!y(c).\overline{c}\mathtt{in}_i(f)!f(g).\overline{g}]^x_y \mapsto^* \overline{x}\mathtt{in}_i$, hence done. (3) follows (2). ■

THEOREM 5.1. (full abstraction) $E \vdash M_1 \cong_\lambda M_2 : T$ *iff* $[\![M_1 : T]\!]_u \cong [\![M_2 : T]\!]_u$.

*Proof.* Let $E \vdash M_{1,2} : T$ with $E = y_1 : T_2, .., y_n : T_n$. By Corollary 5.2, we only have to show the "then" direction. We prove the contrapositive, $[\![M_1 : T]\!]_u \not\cong [\![M_2 : T]\!]_u$ implies $M_1 \not\cong_\lambda M_2$. Below we often omit type annotation for brevity.

$$\begin{array}{ll} \multicolumn{2}{l}{[\![M_1 : T]\!]_u \not\cong [\![M_2 : T]\!]_u} \\ \Leftrightarrow & \exists \vdash_{\mathtt{I}} R \triangleright \overline{E^\circ},\ \vdash_{\mathtt{O}} S \triangleright u:\overline{T^\circ}, v:\mathbb{B}.\quad (\nu x\vec{y})([\![M_1]\!]_u|R|S) \Downarrow^1_v,\ (\nu x\vec{y})([\![M_2]\!]_u|R|S) \Downarrow^2_v \\ & \text{(Lemma 5.1)} \\ \Leftrightarrow & \exists \vdash_{\mathtt{I}} R \triangleright \overline{E^\circ},\ \vdash_{\mathtt{O}} S \triangleright u:\overline{T^\circ}, v:\mathbb{B}. \\ & C_{\mathsf{f}}[(\nu x\vec{y})([\![M_1]\!]_u|R|S)]^w_v \Downarrow_e [\![\mathsf{true}]\!]_w,\ C_{\mathsf{f}}[(\nu x\vec{y})([\![M_2]\!]_u|R|S)]^w_v \Downarrow_e [\![\mathsf{false}]\!]_w. \\ & \text{(Proposition 5.13 (2))} \\ \Leftrightarrow & \exists \vdash_{\mathtt{I}} R \triangleright \overline{E^\circ},\ \vdash_{\mathtt{O}} S \triangleright u:\overline{T^\circ}, v:\mathbb{B}. \\ & (\nu x\vec{y})([\![M_1]\!]_u|R|C_{\mathsf{f}}[S]^w_v) \Downarrow_e [\![\mathsf{true}]\!]_w,\ (\nu x\vec{y})([\![M_2]\!]_u|R|C_{\mathsf{f}}[S]^w_v) \Downarrow_e [\![\mathsf{false}]\!]_w. \\ & \text{(Proposition 5.2 (2-b))} \\ \Rightarrow & \exists. \vdash N_i : T_i,\ u : T \vdash L : \mathbb{B}. \\ & (\nu x\vec{y})([\![M_1]\!]_u|\Pi[\![N_i]\!]_{y_i}|[\![L]\!]_w) \Downarrow_e [\![\mathsf{true}]\!]_w,\ (\nu x\vec{y})([\![M_2]\!]_u|\Pi[\![N_i]\!]_{y_i}|[\![L]\!]_w) \Downarrow_e [\![\mathsf{false}]\!]_w. \\ & \text{(Corollary 5.3)} \\ \Leftrightarrow & \exists. \vdash N_i : T_i,\ u : T \vdash L : \mathbb{B}. \\ & (\lambda u.L)((\lambda y_1..y_n.M_1)\vec{N}) \Downarrow \mathsf{true},\ (\lambda u.L)((\lambda y_1..y_n.M_2)\vec{N}) \Downarrow \mathsf{false}. \\ & \text{(Lemma 5.2)} \\ \stackrel{\text{def}}{\Leftrightarrow} & M_1 \not\cong_\lambda M_2, \end{array}$$

as required. ■

*Remark.* By Corollary 5.3, the embedding is in addition fully complete (in the sense of [3]) up to $\cong$.

## 6. LINEAR π-CALCULUS WITH FREE NAME PASSING

In the previous sections, we have investigated the properties of the linear π-calculus whose outputs are restricted to those which pass only bound names. Using bound names has significance in making the representation of computational behaviour as tight as possible: given some behaviour which we wish to model, the way of representing it in the typed calculus becomes strongly constrained and thus, for example, we own a fairly tractable notion of inhabitants in each type (Theorem 5.1). However, a natural question remains: can we impose behavioural constraints of the similar kind on terms with free name passing, i.e. using the standard syntax for the asynchronous $\pi$-calculus? And if we can, does it add any expressive power? This is not only intellectual curiosity. Apart from the simplicity of the presentation (by moving to free name passing we can get rid of a couple of added structural rules), free name passing makes the computation more tractable: it also has technical advantages in the second-order setting [12].

This section studies these questions, extending the syntax to free outputs while using precisely the same type structures. The typing rules do not change except for free outputs. The embedding of terms of the system with bound outputs into the system with free outputs is essentially subset inclusion. After presenting the translation, we show these two maps not only preserve types but also the semantics: they do not change the behaviour of processes up to the canonical equalities. This result also shows that the universe of linear terms with free outputs is semantically equivalent to its strict subset which use only bound outputs, thus answering the question posed above. The extended reduction is used as a tractable tool to prove their correspondence.

### 6.1. Linear Typing with Free Name Passing

The syntax of processes with free name passing is the standard asynchronous polyadic $\pi$-calculus with branchings and selections. We take off the bound output and selection from the syntax in Section 5 and replace them with the following two.

$$P \quad ::= \quad \ldots \mid \overline{x}\langle\vec{y}\rangle \mid \overline{x}\mathtt{in}_i\langle\vec{y}\rangle$$

The bound output $\overline{x}(\vec{y})P$ can be recovered as $(\nu\,\vec{y})(\overline{x}\langle\vec{y}\rangle|P)$, so that the second syntax in fact subsumes the first one. For the reduction relation $\longrightarrow$, we replace the axioms with:

$$\begin{array}{rcl} x(\vec{y}).P \mid \overline{x}\langle\vec{v}\rangle & \longrightarrow & P\{\vec{v}/\vec{y}\} \\ x[\&_i(\vec{y}_i).P_i] \mid \overline{x}\mathtt{in}_i\langle\vec{v}\rangle & \longrightarrow & P\{\vec{v}/\vec{y}_i\} \end{array}$$

Similarly for replication. The typing rules for these processes are exactly the same except that the sequent is now written $\vdash_{\mathrm{f}} P \triangleright A$ and that we use the following rules for outputs.

$$\text{(Out)}\ \frac{-}{\vdash_{\mathrm{f}} \overline{x}\langle\vec{y}\rangle \triangleright x : (\vec{\tau})^{p_{\mathrm{O}}} \odot \vec{y} : \overline{\vec{\tau}}} \qquad \text{(Sel)}\ \frac{-}{\vdash_{\mathrm{f}} \overline{x}\mathtt{in}_j\langle\vec{y}_j\rangle \triangleright x : [\oplus_i \vec{\tau}_i]^{p_{\mathrm{O}}} \odot \vec{y}_j : \overline{\vec{\tau}_j}}$$

In (Out), we assume $\tau_i = \tau_j$ if $y_i = y_j$. Similarly for (Sel). Note the types for object names in the above two rules are dualised. The resulting system is denoted FNP and the original system is denoted BNP. For clarity we hereafter write $\vdash_{\mathrm{b}} P \triangleright A$ for the typability in BNP.

The rules for outputs given above, are best understood in terms of the following representation of free outputs in the realm of bound outputs.

$$\overline{x}\langle\vec{y}^{\vec{\tau}}\rangle^{\circ} \stackrel{\mathrm{def}}{=} \overline{x}(\vec{w})\Pi_i[w_i \to y_i]^{\tau_i}$$

(the same expression already appeared as $\mathsf{Msg}\langle x\vec{w}\rangle$ in Figure 8). The annotation of free objects do not lose generality since, when processes are typed, we can always restore the original type information. The above representation says that a free name output is a bound name output in which all exported names are "equated" with the mentioned free names. In this representation, $w_i$ is used as $\tau_i$ and, as a result, $y_i$ is used as $\overline{\tau}_i$, illustrating the typing rules given above.

Let $\vdash_{\mathsf{b}} P \triangleright A$ and define $P^\circ$ by extending the above map compositionally, i.e. $\mathbf{0}^\circ \stackrel{\text{def}}{=} \mathbf{0}$, $(P|Q)^\circ \stackrel{\text{def}}{=} P^\circ|Q^\circ$, $((\nu x)P)^\circ \stackrel{\text{def}}{=} (\nu x)(P^\circ)$, $(x(\vec{y}).P)^\circ \stackrel{\text{def}}{=} x(\vec{y}).(P^\circ)$, $(!x(\vec{y}).P)^\circ \stackrel{\text{def}}{=} !x(\vec{y}).(P^\circ)$ and $\overline{x}\langle\vec{y}^{\vec{\tau}}\rangle^\circ \stackrel{\text{def}}{=} \overline{x}(\vec{w})\Pi_i[w_i \to y_i]^{\tau_i}$, similarly for branching. We shall be using this encoding for relating the two systems. We can easily check:

PROPOSITION 6.1. $\vdash_{\mathsf{f}} P \triangleright A$ *iff* $\vdash_{\mathsf{b}} P^\circ \triangleright A$.

While we can directly verify various syntactic properties of $\mathsf{FNP}$, a simple way to do so is by reducing them to those of $\mathsf{BNP}$. We first define the extended reduction for the free output calculus as follows.

$$
\begin{array}{llcl}
(\mathsf{E1_f}) & C[\overline{x}\langle\vec{v}_1\rangle]..[\overline{x}\langle\vec{v}_n\rangle] \mid x(\vec{y}).Q & \mapsto_{\mathsf{l}} & C[Q\{\vec{v}_1/\vec{y}\}]..[Q\{\vec{v}_n/\vec{y}\}] \\
(\mathsf{E2_f}) & C[\overline{x}\langle\vec{v}\rangle] \mid !x(\vec{y}).Q & \mapsto_{\mathsf{r}} & C[Q\{\vec{v}/\vec{y}\}] \mid !x(\vec{y}).Q \\
(\mathsf{E3_f}) & (\nu x)!x(\vec{y}).Q & \mapsto_{\mathsf{g}} & \mathbf{0}
\end{array}
$$

where $(\mathsf{E1_f}, \mathsf{E2_f}, \mathsf{E3_f})$ correspond to $(\mathsf{E1}, \mathsf{E2}, \mathsf{E3})$ respectively (In $(\mathsf{E1_f})$ we incorporate occurrences of linear output names in branches, cf. Section 5.1). $(\mathsf{E1}, \mathsf{E2}, \mathsf{E3})$ are changed accordingly. We now show, via the mapping $(\,\cdot\,)^\circ$, the dynamics of $\mathsf{BNP}$ can completely mimic that of $\mathsf{FNP}$.

PROPOSITION 6.2. (simulation) Let $\vdash_{\mathsf{f}} P \triangleright A$ below.

(1) If $P \equiv Q$ then $P^\circ \equiv Q^\circ$. Also $P^\circ\{\vec{v}/\vec{y}\} \stackrel{\text{def}}{=} (P\{\vec{v}/\vec{y}\})^\circ$.
(2) If $P \longrightarrow Q$ then $P^\circ \longrightarrow\mapsto^* Q^\circ$.
(3) If $P \mapsto Q$ then $P^\circ \mapsto^+ Q^\circ$.

*Proof.* Two statements in (1) are mechanical. For (2) we argue by rule induction. For the base case, we use Proposition 5.4 (2) as well as the latter half of (1) above to obtain:

$$
\begin{array}{rcl}
x(\vec{y}).P|\overline{x}\langle\vec{v}\rangle & \stackrel{\text{def}}{=} & x(\vec{y}).P^\circ|\overline{x}(\vec{y})\Pi[y_i \to v_i] \\
& \longrightarrow & (\nu\vec{y})(P^\circ|[v_i \to y_i]) \\
& \mapsto^* & P^\circ\{\vec{v}/\vec{y}\} \stackrel{\text{def}}{=} (P\{\vec{v}/\vec{y}\})^\circ
\end{array}
$$

hence as required. Similarly for the replicated reduction. The inductive cases are immediate from the corresponding induction hypotheses, using the first part of (1) for the closure under $\equiv$. (3) is similar. ■

Below $\mathsf{CSN}(P)$ and $\mathsf{CSN}_e(P)$ in $\mathsf{FNP}$ are understood as those ideas in $\mathsf{BNP}$.

COROLLARY 6.1.

(1) (reduction) *If* $\vdash_f P \triangleright A$, *then* (i) $P \longrightarrow Q$ *implies* $\vdash_f Q \triangleright A$; (ii) $P \longrightarrow Q_{1,2}$ *implies either* $Q_1 \equiv Q_2$ *or* $Q_{1,2} \longrightarrow R$ *for some* $R$; *and* (iii) $\mathsf{CSN}(P)$.

(2) (extended reduction) *If* $\vdash_f P \triangleright A$, *then* (i) $P \mapsto Q$ *implies* $\vdash_f Q \triangleright A$; (ii) $P \mapsto Q_{1,2}$ *implies* $Q_{1,2} \mapsto^* R$ *for some* $R$; *and* (iii) $\mathsf{CSN}_e(P)$.

*Proof.* Direct from the corresponding results in BNP. As an example, let $\vdash_{\rm f} P \triangleright A$ and $P \longrightarrow Q$. Then $\vdash_{\rm b} P^{\circ} \triangleright A$ by Proposition 6.1. Further let $P \longrightarrow Q$. By Proposition 6.2 we have $P^{\circ} \mapsto^{*} Q^{\circ}$, hence by subject reduction in BNP we have $\vdash_{\rm b} Q^{\circ} \triangleright A$. Again by Proposition 6.1 this means $\vdash_{\rm b} Q \triangleright A$, as required. ■

*Remark. (on bisimilarities)* Define $\approx$ in FNP using the standard free name passing transition, combined with the type-directed constraints given in Figure 4. We can easily show $\approx$ coincides with the congruent closure of $\mapsto$ in FNP, using precisely the same reasoning. This result cannot be obtained via the embedding, because $\approx$ is not abstract enough in comparison with the one induced by the encoding: the equivalence obtained via the embedding (based on $\approx$ in BNP) is strictly more general. We can regain the latter by using a refined typed transition discussed in [12] (also see [14]); though we use neither of these bisimilarities in the following discussions.

*Remark. (encoding)* The encoding of free output into bound name-passing using copy-cat agents was first proposed by Boreale [13]. Later Merro and Sangiorgi proved that his encoding from the (sorted) asynchronous π-calculus into the (sorted) asynchronous localised πI-calculus is fully abstract with respect to a barbed contextual congruence, using non-trivial up-to bisimulation techniques in [47]. The present paper has demonstrated a more direct and simple full-abstraction proof of the same encoding – but strongly typed, using extended reductions $\mapsto$.

### 6.2. Mutual Fully Abstract Embeddings

For mutual embeddings between BNP and FNP, we use the contextual congruences of the previous section (defined by the same clause for BNP and FNP). We write this maximum congruence for BNP and FNP, $\cong_{\rm b}$ and $\cong_{\rm f}$, respectively. If we wish to designate them without specifying which, we write $\cong$. Since the symmetric closure of $\mapsto$ is a typed congruence which respects convergence at $\mathbb{B}$, we know $\mapsto \subset \cong$. We use the following observations [34, 58].

LEMMA 6.1. *For each $\vdash_{\rm b} P \triangleright A$ with $\mathsf{md}(A(x)) \in \{\uparrow, ?\}$ and a fresh name $y$, we have $(\nu x)(P|[x \to y]) \cong_{\rm b} P\{y/x\}$. Similarly, for each $\vdash_{\rm f} P \triangleright A$ with $\mathsf{md}(A(x)) \in \{\uparrow, ?\}$, we have $(\nu x)(P|[x \to y]) \cong_{\rm f} P\{y/x\}$.*

*Proof.* By Proposition 5.4 (2) $(\nu x)(P|[x \to y]) \mapsto^{*} P\{y/x\}$ in BNP. By Proposition 6.2 the same is true for FNP. Since $\mapsto$ stays within $\cong$, we are done. ■

Let $\vdash_{\rm b} P \triangleright A$. Then we write $P^{\star}$ for the result of translating $P$ into a process with free name passing by the following map for bound output, $(\overline{x}(\vec{y})P)^{\star} \stackrel{\rm def}{=} (\nu \vec{y})(\overline{x}\langle\vec{y}\rangle|P^{\star})$, as well as $\mathbf{0}^{\star} \stackrel{\rm def}{=} \mathbf{0}$, $(P|Q)^{\star} \stackrel{\rm def}{=} P^{\star}|Q^{\star}$, $((\nu x)P)^{\star} \stackrel{\rm def}{=} (\nu x)(P^{\star})$, $(x(\vec{y}).P)^{\star} \stackrel{\rm def}{=} x(\vec{y}).(P^{\star})$ and $(!x(\vec{y}).P)^{\star} \stackrel{\rm def}{=} !x(\vec{y}).(P^{\star})$, similarly for branching. We can verify:

PROPOSITION 6.3. $\vdash_{\rm b} P \triangleright A$ *iff* $\vdash_{\rm f} P^{\star} \triangleright A$.

We can now state the main result of this section.

THEOREM 6.1. (both-way retracts) *For each* $\vdash P \triangleright A$*, we have* $P^{\star\circ} \cong_{\mathrm{b}} P$*. Similarly, for each* $\vdash_{\mathrm{f}} P \triangleright A$*, we have* $P^{\circ\star} \cong_{\mathrm{f}} P$*.*

*Proof.* For the first half, we use induction. The only non-trivial case is bound output. Let $\vdash \overline{x}(\vec{y})P \triangleright A$. Then

$$\begin{array}{rcl} (\overline{x}(\vec{y})P)^{\star\circ} & \stackrel{\text{def}}{=} & (\nu\,\vec{y})((\overline{x}(\vec{w})\Pi_i[w_i \to y_i])|P^{\star\circ}) \\ & \equiv & \overline{x}(\vec{w})(\nu\,\vec{y})(\Pi_i[w_i \to y_i]|P^{\star\circ}) \\ & \cong_{\mathrm{b}} & \overline{x}(\vec{y})(P^{\star\circ}) \;\cong_{\mathrm{b}}\; \overline{x}(\vec{y})P \end{array}$$

The last two equations are by Lemma 6.1 and by induction hypothesis, respectively. The second half is also by induction, which boils down to showing $\overline{x}\langle\vec{y}\rangle^{\circ\star} \cong_{\mathrm{f}} \overline{x}\langle\vec{y}\rangle$. In fact, using Lemma 6.1, we have:

$$\overline{x}\langle\vec{y}\rangle^{\circ\star} \stackrel{\text{def}}{=} (\nu\,\vec{w})(\overline{x}\langle\vec{w}\rangle|\Pi_i[w_i \to y_i]) \cong_{\mathrm{f}} \overline{x}\langle\vec{y}\rangle$$

as required. ■

Theorem 6.1 shows that all additional terms in FNP which do not exist in BNP are in fact equivalent to their image in BNP, so that FNP does not add anything to BNP semantically. Further it says that this map is semantically the identity map on BNP. We now conclude the section with a full abstraction result.

COROLLARY 6.2. (full abstraction) $\vdash_{\mathrm{b}} P \cong_{\mathrm{b}} Q \triangleright A$ *iff* $\vdash_{\mathrm{f}} P^{\star} \cong_{\mathrm{f}} Q^{\star} \triangleright A$*. Similarly* $\vdash_{\mathrm{f}} P \cong_{\mathrm{f}} Q \triangleright A$ *iff* $\vdash_{\mathrm{b}} P^{\circ} \cong_{\mathrm{b}} Q^{\circ} \triangleright A$*.*

For the proof we use the characterisation in Proposition 5.2 (2). Suppose $P \cong_{\mathrm{b}} Q$ and $C[P^{\star}] \Downarrow^{i}_{x}$. By Theorem 6.1, $R^{\circ} \cong_{\mathrm{f}} R$, so that $(C[P^{\star}])^{\circ} \Downarrow^{i}_{x}$, that is $C^{\circ}[P^{\star\circ}] \Downarrow^{i}_{x}$. Again by Theorem 6.1 we have $C^{\circ}[P] \Downarrow^{i}_{x}$. By assumption we have $C^{\circ}[Q] \Downarrow^{i}_{x}$, that is $(C^{\circ}[Q])^{\star} \Downarrow^{i}_{x}$, from which we know, again by Theorem 6.1, $C[Q^{\star}] \Downarrow^{i}_{x}$, as required. The converse is trivial since $(\ )^{\star}$ is syntactic identity. The proof of the second half is precisely symmetric.

## 7. DISCUSSION AND FURTHER WORK

**Summary.** The present study is part of our quest to articulate significant classes of computational behaviour using typed π-calculi. Previous work [11] introduced *affine, sequential* types for the π-calculus and established full abstraction for an encoding of PCF, which is the representative sequential functional calculus allowing divergent computation. Using causality between names, the present text refines affine, sequential types into *linear* types to ensure strong normalisability and full abstraction for $\lambda_{\to,\times,+}$. Figure 9 shows the relationship between these results.

- The addition of branching types is indicated by &, $\to$ adds causality to action types, and Seq stands for the inclusion of the sequentiality constraints used in [11].
- Determinacy, SN and sequentiality are properties guaranteed by each typing system.
- FC denotes *full completeness* of the embedding of the corresponding λ-calculus into the π-calculus in the sense of [3] (up to $\equiv$ and η-expansion, cf. Proposition 5.3), while FA stands for *full abstraction* up to semantic equality.

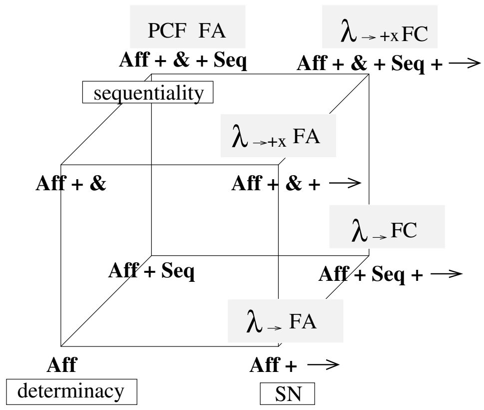


**FIG. 9.** A Family of Affine/Linear Systems

For example, the linear typing system in § 2 corresponds to $\mathsf{Aff} + \rightarrow$, its branching extension in §5 to $\mathsf{Aff} + \& + \rightarrow$ and the sequential system in [11] to $\mathsf{Aff} + \& + \mathsf{Seq}$. Note that the development in § 5 shows that our encoding is already 'almost' fully complete intensionally and indeed becomes fully complete by quotienting with the observational congruence. It is also notable that we could have used the call-by-value encoding in [51] to obtain exactly the same result, indicating the flexibility of the proposed calculus to encode functional SN behaviour. Extensions based on this family are summarised in the introduction, but also in [36, Figure 1] and [69, Figure 1]. See also the discussion below.

**Liveness in Interaction.** A consequence of strong normalisability is liveness in interaction: if a typed agent calls another replicated typed agent and waits for its answer at a linear channel $x$, then an answer is guaranteed to eventually arrive at $x$, however complex intermediate interaction sequences would be.

PROPOSITION 7.1. (linear honesty) *Let* $x:(\tau)^! \in A$ *such that* $\mathsf{md}(\tau) = \uparrow$. *Suppose* $\vdash P \triangleright A$ *with* $A$ *closed. Then* $P \xrightarrow{x(y)} P'$ *implies* $P' \longrightarrow^* \xrightarrow{l}$ *where* $l$ *is an output at* $y$.

For the proof we use the following lemma.

LEMMA 7.1. (linear actions) *Let* $\vdash P \triangleright A, \uparrow x$ *with* $A$ *closed. Then for each* $P \longrightarrow^* P_0$ *there exists* $P'$ *such that* $P_0 \longrightarrow^* P'$ *where* (1) $P'$ *contains no active linear input channels, and* (2) $P' \xrightarrow{l}$ *such that* $l$ *is an output at* $x$.

*Proof.* Without loss of generality assume $P \equiv \Pi_{1..n} P_i^{y_i}$ where each $P_i$ is prime (we can ignore other shapes of $P$ because if there is a restriction at $x$ then the corresponding action type is again closed). By $A$ being closed, all free outputs in $P$ are compensated by free input channels which should be active by well-typedness. This means each $P_i$ is input-prefixed or, if not, it is output-prefixed with $x$. Suppose there is a linear input prefix with subject, say, $y'_1 \in \{y\}$. Since $A$ is closed, there is a compensating output. Since $P \in \mathsf{NF}_e$, this output should be under some input prefix whose subject cannot be $y'_1$ by acyclicity. This means, say, $y'_2 \curvearrowright y'_1$. Again this should be compensated by some output, which should be prefixed by an input with subject $z$, but $z$ cannot be among $\{y'_1, y'_2\}$ by acyclicity again. So set $z = y'_3$. In this way we have a chain of form $y'_n \curvearrowright y'_{n-1} \curvearrowright .. \curvearrowright y'_1$ which exhaust $\{y\}$. But this means $y'_n$ has no compensating output, contradicting $A$'s being closed. Hence there is no active linear input in $P$, establishing (1). Since if $x$ is not active it should be under the prefix of a linear input, this also proves (2), as required.

By CR of $\longrightarrow$ and by Theorem 3.1, this establishes Lemma 7.1. ■

Now Proposition 7.1 is immediate by noting that, after the mentioned input, the term has the action type mentioned in the Lemma after it performs the appropriate input.

We can strengthen Proposition 7.1 by incorporating the possibility that the client itself interacts with the server towards the eventual answer [31]. The central point of the present liveness property is that, in spite of such nested, complex webs of procedure calls, each client is still guaranteed to receive an answer, improving on preceding related type disciplines, cf. [41, 42, 68]. We can further guarantee this liveness property with non-terminated and stateful, non-deterministic computation [69]; this property plays as the key rôle to establish further applications for information flow analysis of programming languages [36, 71].

**State and Non-functional Control.** It is an important subject of study to extend our typing system to allow incorporation of state, control and other non-functional elements. The resulting calculi would be useful as a theoretical basis for the application of SN in a wider realm. Such a formalism might also be useful as a meta-language for logical systems with e.g. non-deterministic cut elimination procedures.

For stateful computation, [69] has verified that our proof method is also applicable in SN for first-order stateful processes combining the proof method established in Classical Logics framework [44, 64]. The basic idea is first replacing replication with recursion [33], then applying the term rewriting technique directly using the extended reduction. This allows to carry over the SN type discipline and related results in imperative computation involving non-trivial procedure calls in [36].

For the incorporation of control into the present type discipline, all we need is to eliminate $\downarrow$-$\uparrow$ types from the present system. In other words, the system presented in Section 2 already contains the calculus for (linear) control as its subcalculus. This means, among others, the calculus satisfies all syntactic properties we have explored in Sections 2 and 3, including strong normalisability. We have verified that a sequential version of this calculus can fully abstractly embed Parigo's $\lambda\mu$-calculus. Further discussions on this calculus and its extensions will be discussed elsewhere.

**Second-order and Other Extensions in Type Structure.** Can the presented results be augmented to cover more expressive notions of types studied in functional calculi? Adding recursive types [52, 66] easily leads to a system that is not strongly normalising: for example, the encoding, following Figure 8, of $(\lambda x.xx)(\lambda x.xx)$ becomes typable. Regarding second-order types, our recent work [12] demonstrates that such extensions coexist harmoniously with SN, as they do in the corresponding functional calculi. In particular, the causality constraints formalised in the present paper are sufficient to encode System F fully abstractly in the second-order extension of the present system. Other, more refined type structures would also be worth studying in the present context: the $\pi$-calculus offers a natural habitat to SN typing systems for stateful, control [30], interactive and mobile computation.

**Complex Causality.** The present work adds minimum causality to the system in [11] to ensure SN of replicated processes. However, our SN proof seems to be able to cope, without significant change, with more complex causality relations: for example, we could relax the channel type constraints and extend action types to finite graph structures between arbitrary linear nodes as in [68]. An even wider class of SN interactions would be typable if we further allowed edges of the more general form $px \to qy$, where $p \in \{\downarrow, \uparrow, ?\}$ and $q \in \{!, \downarrow, \uparrow\}$ (i.e. replicated and linear nodes can be mixed). Diverse structures would be embeddable in such an extension, including full proof nets [9]. The status of strong reduction would become subtle in this setting, cf. [23].

**Game Semantics.** In game semantics, "winning strategies" represent strong normalisation [3]. This representation ensures, essentially by definition, that composition of two winning strategies will never go into infinite $\tau$-actions (which would make the strategy partial). This extensional representation of SN does not directly suggest concrete type disciplines to ensure SN for mobile processes, even though the liveness property discussed in Proposition 7.1 closely corresponds to the games-based characterisation of SN. In this context, we observe that the sequential version of the linear $\pi$-calculus discussed in Section 5.3 is the linear refinement of the affine sequential calculus in [10, 11]. This immediately shows that the typed sequential transition for the linear sequential $\pi$-calculus is *innocent* in the sense of [4, 10, 38]. The linear liveness in Proposition 7.1 further indicates that it is *total*, in the sense that it is always defined for each legal input; and, moreover, we can show it is *finite* in that the cardinality of the induced innocent function for each process is finite. It would be interesting to use the framework introduced in the present paper, among others typed processes and their behavioural characterisation, for formulating and studying various notions of SN and related ideas in game semantics (for example we may consider explicit incorporation of acyclicity conditions).

**Term Rewriting and Reduction Strategies.** The proof method presented in this paper uses the extended reduction $\mapsto$ to prove not only SN but also other results including a fully abstract embedding of $\lambda_{\to,\times,+}$. One of the merits in using $\mapsto$ lies in the potential applicability of various Term Rewriting Lemmas in the context of interacting processes. In fact, technically speaking, this may be regarded as one of the main differences from other studies addressing termination and other related properties of processes [42, 60]. Our recent work [69] partly addresses this point. In Section 3, we define a reduction strategy of $\mapsto$ to prove SN. Like the left-most reduction strategy of the $\lambda$-calculus, this strategy could be defined in the untyped setting in general, then could be used to prove the normalisation theorem (i.e. it always derives a normal form if it exists). This opens possibility to study various reduction strategies in the name-passing scenario, which had not been investigated so far due to, among others, existence of structure rules. We may hope that, through such studies, that the accumulated ideas from functional computation such as optimality [46] may be transferable into non-deterministic and non-terminating interactive computation.

## REFERENCES


1. Abramsky, S., Computational interpretation of linear logic. *TCS*, Vol. 111 (1993) 3–57, 1993.
2. Abramsky, S., Proofs as Processes, *TCS*, Vol. 135 (1994) 5–9, 1994.
3. Abramsky, S. and Jagadeesan, R., Games and Full Completeness for Multiplicative Linear Logic. *JSL*, Vol. 59, 1994.

4. Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF, *Info. & Comp.* 163 (2000), 409-470.
5. Abramsky, S., Process Realizability, A Tutorial Workshop on Realizability Semantics and Applications, 1999. Available at web.comlab.ox.ac.uk/oucl/work/samson.abramsky.
6. Altenkirch, T., Dybjer, P., Hofmann, M. and Scott, P., Normalisation by Evaluation for Typed Lambda Calculus with Coproducts, *LICS'01* 303–310, IEEE, 2001.
7. Barendregt, H., *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.
8. Barendregt, H., Lambda Calculi with Types, Handbook of Logic in Computer Science, Vol 2, pp.118–310, Clarendon Press, Oxford, 1992.
9. Bellin, G., and Scott, P.J., On the Pi-calculus and linear logic, *TCS*, Vol. 135, pp. 11-65, 1994.
10. Berger, M., Honda, K. and Yoshida, N., Sequentiality and the π-Calculus, *TLCA01*, LNCS 2044, 29–45, Springer, 2001.
11. The full version of [10]. Available at www.dcs.qmw.ac.uk/˜kohei.
12. Berger, M., Honda, K. and Yoshida, N., Genericity and the π-Calculus, *FoSSaCs03*, LNCS 2620, 103–119, Springer, 2003.
13. Boreale, M., On expressiveness in internal mobility of name passing calculi, *TCS*, 195:206–226, 1998.
14. Boreale, M. and Sangiorgi, D., Some Congruence Properties for π-calculus Bisimilarities, *TCS*, 1998, Vol.198, pages = 159–176.
15. Boudol, G., Asynchrony and the pi-calculus, INRIA Research Report 1702, 1992.
16. Boudol, G., The pi-calculus in direct style, *POPL'97*, 228–241, ACM, 1997.
17. Boudol, G. and Castellani, I., Noninterference for Concurrent Programs, *ICALP01*, LNCS 2076, 382–395, Springer, 2001.
18. Chaki, S., Rajamani, S. and Rehof, J., Types as Models: Model Checking Message-Passing Programs, *POPL'02*, ACM, 2002.
19. Ghani, N., *Adjoint Rewriting*, PhD Thesis, LFCS, University of Edinburgh, Nov. 1995.
20. Ghani, N., βη-equality from coproducts, *TLCA'95*, LNCS 902, 171–185, Springer, 1995.
21. Gallier, J. H., On Girard's "Candidats de Reductibilité", 123–203, Logic and Computer Science, Academic Press Limited, 1990.
22. Gay, S. and Hole, M., Types and Subtypes for Client-Server Interactions, *ESOP'99*, LNCS 1576, 74–90, Springer, 1999.
23. Girard, J.-Y., Linear Logic, *TCS*, Vol. 50, 1–102, 1987.
24. Girard, J.-Y., Lafont Y. and Taylor, P., *Proofs and Types*, vol. 7 of Cambridge Tracts in Theoretical Computer Science, CUP, 1989.
25. Gunter, C.A., *Semantics of Programming Languages: Structures and Techniques*, MIT Press, 1992.
26. Hicks, M., Kakkar, P., Moore, J.T., Gunter, C.A. and Nettles, S. PLAN: A Packet Language for Active Networks, *Proceedings of the International Conference on Functional Programming (ICFP'98)*, cf. [56].
27. Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, 509-523, 1993.
28. Honda, K., Composing Processes, *POPL'96*, 344-357, ACM, 1996.
29. Honda, K., Notes on Linear Typing for Free Outputs, May, 2001.
30. Honda, K., Notes on the linear π-calculus and LLP, June, 2001.
31. 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.
32. Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, 133–147, Springer-Verlag 1991.
33. Honda, K. Vasconcelos, V., and Yoshida, N. Secure Information Flow as Typed Process Behaviour, *ESOP '99*, LNCS 1782, 180–199, Springer-Verlag, 2000. Full version available as MCS technical report 01/2000, University of Leicester, 2000.
34. Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *TCS*, 437–486, Vol. 151, North-Holland, 1995.
35. Honda, K. and Yoshida, N. Game-theoretic analysis of call-by-value computation. *TCS* Vol. 221 (1999), 393–456, North-Holland, 1999.

36. Honda, K. and Yoshida, N., A Uniform Type Structure for Secure Information Flow, *POPL'02*, 81–92, ACM Press, 2002. Full version as a DOC technical report, Department of Computing, Imperial College London, August, 2002. Available at: www.doc.ic.ac.uk/˜yoshida.
37. Huet, G., Confluent reductions: Abstract properties and applications to term rewriting systems, *J., Assoc, Comput. March*, 23(1), 1981, 11-21.
38. Hyland, M. and Ong, L., "On Full Abstraction for PCF": I, II and III. *Info. & Comp.* 163 (2000), 285-408.
39. Igarashi, A. and Kobayashi, N., A generic type system for the pi-calculus, *POPL'01*, ACM, 2001.
40. Klop, J. W., Term Rewriting Systems, Handbook of Logic in Computer Science, Vol 2, pp.2–117, Clarendon Press, Oxford, 1992.
41. Kobayashi, N., A partially deadlock-free typed process calculus, *ACM TOPLAS*, Vol. 20, No. 2, 436–482, 1998.
42. Kobayashi, N., Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness, *Proc. of TCS2000*, LNCS 1872, pp.365-389, Springer, 2000.
43. Kobayashi, N., Pierce, B., and Turner, D., Linear Types and π-calculus, *POPL'96*, 358–371, ACM Press, 1996.
44. Laird, J., A deconstruction of non-deterministic classical cut elimination, *TLCA'01*, LNCS 2044, 268–282, Springer, 2001.
45. Lafont, Y., Interaction Nets, *POPL'90*, pp. 95–108, ACM press, 1990.
46. Lévy, J.-J., An algebraic interpretation of the λ-β-*K*-calculus and a labelled λ-calculus, pp.147–165, Proc. of *λ-calculus and Computer Science Theory*, LNCS 37, Springer-Verlag, 1975.
47. Merro, M. and Sangiorgi, D., On asynchrony in name-passing calculi, *ICALP'98*, LNCS 1443, 856–867, Springer-Verlag, 1998.
48. Milner, R., *A Calculus of Communicating Systems*, LNCS 76, Springer-Verlag, 1980.
49. Laurent, O., Polarized games, LICS 2002, 265-274, IEEE, 2002.
50. Laurent, O., Invited lecture: Polarities in linear logic. Linear Logic Workshop, July 2002.
51. Milner, R., Functions as Processes. *MSCS*, 2(2), 119–146, CUP, 1992.
52. Milner, R., Polyadic π-Calculus: a tutorial. *Proceedings of the International Summer School on Logic Algebra of Specification*, Marktoberdorf, 1992.
53. Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), 1–77, 1992.
54. Mitchell, J. *Foundations for Programming Languages*, MIT Press, 1996.
55. Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *LICS'93*, 187–215, IEEE, 1993.
56. PLAN: A Packet Language for Active Networks, SwitchWare Project, University of Pennsylvania, available from http://www.cis.upenn.edu/˜switchware/.
57. Quaglia, P. and Walker, D., On Synchronous and Asynchronous Mobile Processes, *FoSSaCS 00*, LNCS 1784, 283–296, Springer, 2000.
58. Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, North-Holland, 1996.
59. Sangiorgi, D., The name discipline of uniform receptiveness, *ICALP'97*, LNCS 1256, 303–313, Springer, 1997.
60. Sangiorgi, D., Termination of Processes, Draft, December, 2001.
61. Smith, G., A New Type System for Secure Information Flow, *CSFW'01*, IEEE, 2001.
62. Smith, G. and Volpano, D., Secure information flow in a multi-threaded imperative language, 355–364, *POPL'98*, ACM, 1998.
63. Tait, W., Intensional interpretation of functionals of finite type, I. *J. Symb. Log*, 32, 198–212, 1967.
64. Urban, C. and Bierman, G.M., Strong Normalisation of Cut-Elimination in Classical Logic. *Fundamenta Informaticae*, 45(1-2):123–155, 2001.
65. Vasconcelos, V., Typed concurrent objects. *ECOOP'94*, LNCS 821, 100–117. Springer, 1994.
66. Vasconcelos, V. and Honda, K., Principal Typing Scheme for Polyadic π-Calculus. *CONCUR'93*, LNCS 715, 524–538, Springer-Verlag, 1993.
67. Winskel, G., *The Formal Semantics of Programming Languages: An Introduction*, MIT Press, 1993.

68. Yoshida, N., Graph Types for Monadic Mobile Processes, *FST/TCS'16*, LNCS 1180, 371–387, Springer-Verlag, 1996. Full version as LFCS Technical Report, ECS-LFCS-96-350, 1996.

69. Yoshida, N., Type-Based Liveness Guarantee in the Presence of Nontermination and Nondeterminism, April 2002. MCS Technical Report, 2002-20, University of Leicester. Available at www.doc.ic.ac.uk/˜yoshida.

70. Yoshida, N., Berger, M. and Honda, K., Strong Normalisation in the π-Calculus, *LICS'01*, 311–322, IEEE, 2001. The first full version as MCS technical report, 2001-09, University of Leicester, 2001.

71. Yoshida, N., Honda, K. and Berger, M. Linearity and Bisimulation, Proc. of 5th International Conference, Foundations of Software Science and Computer Structures (FoSSaCs 2002), LNCS 2303, pp.417–433, Springer, 2002. A full version as a MCS technical report, 2001-48, University of Leicester, 2001. Available at www.doc.ic.ac.uk/˜yoshida.

# APPENDIX A

## A.1. PROOFS FOR SECTION 2

## A.2. PROOF OF SUBJECT REDUCTION THEOREM

We prove Proposition 2.2 (1). The key point is to prove basic properties of algebra on action types. We use the same routine as in [11, 33, 68]. We first show:

LEMMA A.1. *Assume* $A, A_1$ *and* $A_2$ *are action types.*

(1) $A/\vec{y}$ *is an action type.*

(2) *If* $A_1 \asymp A_2$*, then* $A_1 \odot A_2$ *is an action type.*

*Proof.* As the proof in Lemma 3.4 in [33]. (1) is trivial. The case $\mathsf{fn}(A_1) \cap \mathsf{fn}(A_2) = \emptyset$ in (2) is obvious. The other case is proved by induction on the size of $A_1$ and $A_2$ using the BNF representation of action types. ■

LEMMA A.2. *Let* $A_1$*,* $A_2$*,* $A_3$ *be action types. Then we have:*

(1) (commutativity) *Assume* $A_1 \asymp A_2$*. Then we have* $A_2 \asymp A_1$ *and* $A_1 \odot A_2 = A_2 \odot A_1$.

(2) (associativity) *Assume* $A_1 \asymp A_2$ *and* $(A_1 \odot A_2) \asymp A_3$*. Then we have:* (1) $A_1 \asymp A_3$ *and* $A_2 \asymp A_3$*,* (2) $A_1 \asymp (A_2 \odot A_3)$ *and* (3) $(A_1 \odot A_2) \odot A_3 = A_1 \odot (A_2 \odot A_3)$.

*Proof.* (1) is obvious by definition. (2) is proved by induction on the size of $A_i$ using the BNF representation of action types. This is proved as (a special case of) Lemma 3.5 in [33]. ■

LEMMA A.3.

(1) *If* $x\!:\!\tau \in |A|$ *and* $\mathsf{md}(\tau) \in \{!, \downarrow\}$ *then there is no* $y\!:\!\tau' \in |A|$ *such that* $y\!:\!\tau' \to x\!:\!\tau$.

(2) $?B \asymp ?B$ *and* $B \odot B = B$.

(3) *If* $A \asymp B$ *with* $A/\vec{x} = A_0$*,* $x_i\!:\!\tau_i \in |A|$, $\mathsf{md}(\tau_i) \in \{!, \downarrow, \updownarrow\}$*, and* $\mathsf{fn}(B) \cap \{\vec{x}\} = \emptyset$*, then* $A_0 \asymp B$ *and* $(A \odot B)/\vec{x} = A_0 \odot B$.

(4) *If* $A \asymp B$ *with* $A/\vec{x} = A_0$*,* $x_i\!:\!\tau_i \in |A|$, $\mathsf{md}(\tau_i) \in \{!, \downarrow, \updownarrow\}$*, and* $B/\vec{x} = B_0$*, then* $A_0 \asymp B$*,* $A \asymp B_0$*,* $A_0 \asymp B_0$*, and* $(A \odot B)/\vec{x} = A_0 \odot B_0$.

(5) *Suppose* $A/\vec{x} = A_0$*,* $B/\vec{x} = B_0$ *and* $A_0 \asymp B_0$*. Assume* $x_i : \tau_i \in |A|$ *with* $\mathsf{md}(\tau_i) \in \{\updownarrow, !, \downarrow\}$*, and if* $x_i\!:\!\tau_i' \in |B|$*, then* $\tau_i \asymp \tau_i'$*. Then* $A \asymp B$ *and* $(A \odot B)/\vec{x} = A_0 \odot B_0$.

*Proof.* (1) is by the definition of permissibility of $\rightarrow$, i.e. there is no edge to inputs and $\updownarrow$. (2) is obvious by $\tau \odot \tau = \tau$ with $\mathsf{md}(\tau) = ?$. For (3), by (1), we can write $A = \odot_i(x_i : \tau_i \rightarrow A_i'), A'$ since $\mathsf{md}(\tau_i) \in \{!, \downarrow, \updownarrow\}$ (note $A_i$ may be $\emptyset$). Then by $A \asymp B$, obviously $A_0 = (A_i, A') \asymp B$. Hence we have $(A \odot B)/\vec{x} = (A/\vec{x} \odot B/\vec{x}) = A_0 \odot B$. The proof of (4) is similar. (5) uses (2) and (4). ■

*Remark.* If we delete the side condition $\mathsf{md}(\tau_i) \in \{!, \downarrow, \updownarrow\}$ in (5), the property does not hold. For counterexample, let $A = x_1 : \tau_1 \rightarrow x_2 : \overline{\tau_2}$ and $B = x_2 : \tau_2 \rightarrow x_1 : \overline{\tau_1}$. Then $A/x_1x_2 \asymp B/x_1x_2$, but $A \odot B$ is undefined.

LEMMA A.4.

(1) $\vdash P \triangleright A$ *and* $P \equiv Q$ *then* $\vdash Q \triangleright A$.

(2) $\vdash x(\vec{y}{:}\vec{\tau}).P \,|\, \overline{x}(\vec{y}{:}\vec{\tau}')Q \triangleright A$ *implies* $\vdash (\nu\, \vec{y}{:}\vec{\tau}'')(P \,|\, Q) \triangleright A$ *with* $\tau_i' = \overline{\tau_i}$ *and* $\tau_i'' = \tau_i \odot \tau_i'$.

(3) $\vdash !x(\vec{y}:\vec{\tau}).P \,|\, \overline{x}(\vec{y}:\vec{\tau}')Q \triangleright A$ *implies* $\vdash !x(\vec{y}:\vec{\tau}).P \,|\, (\nu\, \vec{y}:\vec{\tau}'')(P \,|\, Q) \triangleright A$ $\tau_i' = \overline{\tau_i}$ *and* $\tau_i'' = \tau_i \odot \tau_i'$.

*Proof.* The proof is essentially the same as in [11, 33]. Assume $\vdash P \triangleright A$. Then, as in [11, Proposition 1 (ii)], there exists a minimum action type $A_0$ such that $A_0 \subseteq A_1$ and $\vdash P \triangleright A_0$ (since we only have to use (Weak) before restriction and input rules). Hence in the following we only consider the minimum action types.

**(1)** By rule induction on $\equiv$. The case of $P \,|\, \mathbf{0} \equiv P$ is easy because $\emptyset$ is a unit of $\odot$. Similarly the cases of $P \,|\, Q \equiv Q \,|\, P$, and $(P \,|\, Q) \,|\, R \equiv P \,|\, (Q \,|\, R)$ are proved by Lemma A.2 (1) and (2), respectively. Next take the structural rule

$$(\overline{x}(\vec{y})P) \,|\, Q \;\equiv\; \overline{x}(\vec{y})(P|Q) \quad \text{with } x \notin \mathsf{fn}(Q)$$

Assume $\vdash (\overline{x}(\vec{y})P)|Q \triangleright A$ such that $x \notin \mathsf{fn}(Q)$. By typing rules we can assume $\vdash \overline{x}(\vec{y})P \triangleright A_1$ and $\vdash Q \triangleright A_2$ with $A_1 \asymp A_2$ and $A_1 \odot A_2 = A$. By strengthening of bases we can set $\{\vec{y}\} \cap \mathsf{fn}(A_2) = \emptyset$. From $\vdash \overline{x}(\vec{y})P \triangleright A_1$ we deduce $\vdash P \triangleright A_1', \vec{y}{:}\vec{\tau}$ with $A_1 = A_1' \odot x{:}(\vec{\tau})^{p_O}$. By Lemma A.3 (5) and associativity, we have $(A_1', \vec{y}{:}\vec{\tau}) \asymp A_2$ and $((A_1', \vec{y}{:}\vec{\tau}) \odot A_2)/\vec{y} \odot x{:}(\vec{\tau})^{p_O} = A$, so that $\vdash \overline{x}(\vec{y})(P|Q) \triangleright A$. The inverse and other cases are similarly dealt with.

**(2)** It is proved by the same reasoning as **(3)** below (the proof is simpler than **(3)**).

**(3)** We prove the monadic case. The polyadic case is just the same. Suppose

$$\vdash !x(y:\tau).P \,|\, \overline{x}(y:\tau')Q \triangleright A$$

Then, we have the derivations such that

$$\vdash !x(y:\tau).P \triangleright A_1 \quad \text{and} \quad \vdash \overline{x}(y:\tau')Q \triangleright A_2$$

with $A_1 \stackrel{\text{def}}{=} (x:(\tau)^! \rightarrow ?A_1')$ and $A_1 \odot A_2 = A$. Then the above input and output processes are derived by ($\mathsf{In}^!$) and ($\mathsf{Out}$) from

$$\vdash P \triangleright y{:}\tau, A_1' \;\text{ and }\; \vdash Q \triangleright (y{:}\tau' \rightarrow A_2'), A_2'' \;\text{ with }\; (A_2', A_2'') \odot x{:}(\tau')^? = A_2$$

First, by $A_1 \asymp A_2$, we have $\tau = \overline{\tau'}$. Also by (iv), we have $A_1' \asymp (A_2', A_2'')$ and $A_1' \odot (A_2', A_2'') = A_1/x \odot A_2/x$.

Suppose $\mathsf{md}(\tau) = \downarrow$. Then we have

$$\vdash P \,|\, Q \triangleright y\!:\!\updownarrow, (A_1/x \odot A_2/x)$$

Hence by (Res), we have:

$$\vdash (\nu y\!:\!\updownarrow)(P \,|\, Q) \triangleright (A_1/x \odot A_2/x)$$

Next we apply (Weak) to $A_2/x$ in order to obtain $A_2$. Then we have:

$$\vdash (\nu y\!:\!\updownarrow)(P \,|\, Q) \triangleright A_1/x \odot A_2$$

By (4) in Lemma A.3, $A_1 \odot A_1/x = A_1$, together with associativity, we finally have

$$\vdash !x(y\!:\!\tau).P \,|\, (\nu y\!:\!\updownarrow)(P \,|\, Q) \triangleright A$$

The case $\mathsf{md}(\tau) = ?$ is just the same by replacing $\updownarrow$ by $\overline{\tau}$ above. ∎

By the above lemma, we conclude with Proposition 2.2 (1).

### A.3. PROOF OF STRONGLY CONFLUENCE

We prove Proposition 2.2 (ii). The only case for a critical pair is $\langle !a(\vec{x}).P,\ \overline{a}(\vec{x})Q_1\rangle$ and $\langle !a(\vec{x}).P,\ \overline{a}(\vec{x})Q_2\rangle$. If $R$ contains this critical pair, then we can write down

$$R \equiv (\nu\,\vec{c})(!a(\vec{x}).P \,|\, \overline{a}(\vec{x})Q_1 \,|\, \overline{a}(\vec{x})Q_2 \,|\, Q')$$

Suppose

$$\begin{array}{l} R \longrightarrow (\nu\,\vec{c})(!a(\vec{x}).P \,|\, (\nu\,\vec{x})(P \,|\, Q_1) \,|\, \overline{a}(\vec{x})Q_2 \,|\, Q') \stackrel{\text{def}}{=} R_1 \quad \text{and} \\ R \longrightarrow (\nu\,\vec{c})(!a(\vec{x}).P \,|\, \overline{a}(\vec{x})Q_1 \,|\, (\nu\,\vec{x})(P \,|\, Q_2) \,|\, Q') \stackrel{\text{def}}{=} R_2. \end{array}$$

Then by contracting $!a(\vec{x}).P$ and $\overline{a}(\vec{x})Q_2$ in $R_1$, we have

$$R_1 \longrightarrow (\nu\,\vec{c})(!a(\vec{x}).P \,|\, (\nu\,\vec{x})(P \,|\, Q_1) \,|\, (\nu\,\vec{x})(P \,|\, Q_2) \,|\, Q') \stackrel{\text{def}}{=} P'$$

Now by contracting $!a(\vec{x}).P$ and $\overline{a}(\vec{x})Q_1$ in $R_2$, we have $R_2 \longrightarrow P'$. ∎

## APPENDIX B

## PROOFS FOR SECTION 3

### B.1. PROOF OF PROPOSITION 3.1

(1) is essentially identical with the proof of $\longrightarrow$, using Lemma A.4. For (2), first we start from the following lemma about garbage collection and linear reduction. The proof is mechanical. We assume all terms are typable.

LEMMA B.1.

(1) (postponement of $\mapsto_{\mathsf{g}}$) *If* $P \mapsto_{\mathsf{g}} Q \mapsto_{\mathsf{l}} R$, *then for some* $S$, $P \mapsto_{\mathsf{l}} S \mapsto_{\mathsf{g}} R$. *Similarly if* $P \mapsto_{\mathsf{g}} Q \mapsto_{\mathsf{r}} R$, *then for some* $S$, $P \mapsto_{\mathsf{r}} S \mapsto_{\mathsf{g}}^{+} R$.

(2) (strong confluence of $\mapsto_{\mathsf{g}}$) *If* $P \mapsto_{\mathsf{g}} Q_i$ $(i = 1,2)$, *then* $Q_1 \equiv Q_2$ *or there exists* $R$ *such that* $Q_i \mapsto_{\mathsf{g}} R$.

(3) (strong normalisation of $\mapsto_{\mathsf{g}}$) *For all* $P$, *there exists* $Q$ *such that* $P \mapsto_{\mathsf{g}}^{*} Q$ *and* $Q \not\mapsto_{\mathsf{g}}$.

(4) (strong confluence of $\mapsto_{\mathsf{l}}$) *If* $P \mapsto_{\mathsf{l}} Q_i$ $(i = 1,2)$, *then* $Q_1 \equiv Q_2$ *or there exists* $R$ *such that* $Q_i \mapsto_{\mathsf{l}} R$.

(5) (strong normalisation of $\mapsto_{\mathsf{l}}$) *For all* $P$, *there exists* $Q$ *such that* $P \mapsto_{\mathsf{l}}^{*} Q$ *and* $Q \not\mapsto_{\mathsf{l}}$.

Let us define $\mapsto_0 \stackrel{\text{def}}{=} (\mapsto_{\mathsf{r}} \cup \mapsto_{\mathsf{l}})$. By postponement of $\mapsto_{\mathsf{g}}$, if $P \mapsto^{*} R$, then there exists $S$ such that $P \mapsto_0^{*} S \mapsto_{\mathsf{g}}^{*} R$. Since $\mapsto_{\mathsf{g}}$ always canonically terminates, we only have to show the CR of $\mapsto_0$ (cf. [40]). For this, it is sufficient to show the following *strip* lemma.

LEMMA B.2. (strip) *If* $P \mapsto_0 P_1$ *and* $P \mapsto_0^{*} P_2$, *then there exists* $P_3$ *such that* $P_1 \mapsto_0^{*} P_3$ *and* $P_2 \mapsto_0^{*} P_3$.

*Proof.* The only interesting case is that an uncontructed message appears under a replicated input. The proof we use here is similar to the one used in Chapter 11 Section 1 of [7] based on the labelled reduction [46]. Our case is simpler since we only contract one message at each step and there is no overlap of occurrences of two messages which are duplicated from the same replication (cf. [46] and Section 11.2 in [7]). Let $P_1$ be the result of replacing the redex $\overline{a}(\vec{y})Q_1$ in $P$ by its reduct $(\nu\,\vec{y})(R_1 \mid Q_1)$. If we keep track of what happens with $\overline{a}(\vec{y})Q_1$ during the reduction $P \mapsto_0^{*} P_2$, then we can find $P_3$. To be able to trace $\overline{a}(\vec{y})Q_1$, we define a new set of terms where uncontructed messages can appear underlined [46]. Consequently, if we underline $a$ in $\overline{\underline{a}}(\vec{y})Q_1$, we only need to reduce all occurrences of the underlined $\underline{a}$ in $P_2$ to obtain $P_3$. The rest is the just same as in [7]. ■

By Lemma B.2, we obtained CR-property of $\mapsto$ (Proposition 3.1 (2)). To prove that the first statement in Proposition 3.1 (3), we note that $P \mapsto_{\mathsf{g}} P'$ and $\mathsf{SN}_e(P')$ does not normally imply $\mathsf{SN}_e(P)$ in untyped setting (e.g. $Q \Uparrow_e$ but $(\nu\,x)!x(\vec{y}).Q \mapsto_{\mathsf{g}} \mathbf{0}$). Hence we shall prove this statement using postponement of $\mapsto_{\mathsf{g}}$, Lemma B.1 (1).

LEMMA B.3.

(1) *If* $P \mapsto_0 P'$ *and* $\mathsf{SN}_e(P')$, *then* $\mathsf{SN}_e(P)$.
(2) *Suppose* $P \not\mapsto_0$. *Then* $P \mapsto_{\mathsf{g}} P'$ *and* $\mathsf{SN}_e(P')$ *implies* $\mathsf{SN}_e(P)$.
(3) *If* $P \mapsto P'$ *and* $\mathsf{SN}_e(P')$, *then* $\mathsf{SN}_e(P)$.

*Proof.* For (1), we can easily check $P \mapsto_0 P_i$ $(i = 1,2)$ with $P_1 \equiv P_2$ implies that there exists $R$ such that $P_i \mapsto_0^{+} R$. Then the rest is standard with Lemma B.2, cf. [1, 7, 37, 40]. (2) is by strong normalisation and church-rosser of $\mapsto_{\mathsf{g}}$. For (3), by (1) in this lemma, we only have to prove $P \mapsto_{\mathsf{g}} P'$ and $\mathsf{SN}_e(P')$ implies $\mathsf{SN}_e(P)$. Then by Lemma B.1 (1) there exists at least one pass such that $P' \mapsto_0^{*} P_1 \mapsto_{\mathsf{g}}^{*} R \not\mapsto$ with $P_1 \not\mapsto_0$. Since $\mathsf{SN}_e(R)$, we have $\mathsf{SN}_e(P_1)$ by (2). Now by applying Lemma B.1 (1) again, we have some $P_1'$ such that $P \mapsto_0^{*} P_1' \mapsto_{\mathsf{g}}^{*} P_1 \mapsto_{\mathsf{g}}^{*} R \not\mapsto$ with $P_1' \not\mapsto_0$. We again have $\mathsf{SN}_e(P_1')$ by (2), from which we can obtain $\mathsf{SN}_e(P)$ by (1), as required. ■

The rest of Proposition 3.1 (3) is straightforward by this and CR property of $\mapsto$.

# APPENDIX C

## PROOFS FOR SECTION 4

### C.1. PROOF OF PROPOSITION 4.2

By induction on generation rules of $\longrightarrow$, it is easy to check $P \longrightarrow Q$ implies $P^A \xrightarrow{\tau} Q^A$. For the other direction, we first show, by rule induction on transition rules, that, if $x$ has mode $\downarrow$ (resp. !), $P^A \xrightarrow{x(\vec{y})} Q^A$ implies $P \equiv C[x(\vec{y}).P_1]$ and $Q \equiv C[P_1]$ (resp. $P \equiv C[!x(\vec{y}).P_1]$ and $Q \equiv C[!x(\vec{y}).P_1|P_1]$) where $C[\;]$ is a reduction context. Similarly for $P^A \xrightarrow{\overline{x}(\vec{y})} Q^A$. Using them we show, again by rule induction on transition rules, that $P^A \xrightarrow{\tau} Q^A$ implies $P \equiv C[C_1[(!)x(\vec{y}).P_1][C_2[\overline{x}(\vec{y})P_2]]$ where $C$, $C_1$ and $C_2$ are reduction contexts. From this it is immediate $P^A \xrightarrow{\tau} Q^A$ implies $P \longrightarrow Q$. ■

### C.2. PROOF OF PROPOSITION 4.3

For (1), we show that the added rules are derivable from the rules in Figure 4, which is immediate. For (2) we first show, in the syntactic transition, if $P_0^A \xrightarrow{l} Q_0^B$ and $P \equiv P_0$ then $P^A \xrightarrow{l} Q^B$ such that $Q \equiv Q_0$. The proof is standard, using rule induction on the syntactic transition system together with inspection of the structure of processes. From this it is easy to check that the given statement holds, this time by rule induction on the original transition system. Finally for (3) "if" is by (1) while "only if" is by (2), noting, under Convention 3.1, the transition induced by syntactic transition system and the one induced by prime syntactic transition system is identical. ■

### C.3. PROOF OF PROPOSITION 4.4

Using the characterisation in Proposition 4.3 (1)(2), it is enough to show $\approx$ derived using the syntactic transition is a typed congruence. Input prefixes, parallel composition and restriction are entirely standard, cf. [48]. For output prefix we define $\mathbf{R} \stackrel{\text{def}}{=} \mathbf{R}_1 \cup \mathbf{R}_2 \cup \mathbf{R}_3$ where:

(1) $\mathbf{R}_1 \stackrel{\text{def}}{=} \approx$;
(2) $\mathbf{R}_2 \stackrel{\text{def}}{=} \{\langle \overline{x}(\vec{y})P_1, \overline{x}(\vec{y})P_2 \rangle \mid P_1 \approx P_2\}$; and
(3) $\mathbf{R}_3 \stackrel{\text{def}}{=} \{\langle (\nu\,\vec{y})P_1\{\vec{y}/\vec{z}\}, (\nu\,\vec{y})P_2\{\vec{y}/\vec{z}\} \rangle \mid P_1 \approx P_2\}$.

In (3) we assume the mentioned substitution is well-typed. It is easy to check whenever $P\mathbf{R}_1 \cup \mathbf{R}_2 Q$ its derivatives are related by $\mathbf{R}$. For $\mathbf{R}_3$ we show this relation coincides with $\approx$. Clearly $\mathbf{R}_3 \supset \approx$ (let $\vec{y}$ be the empty string). For the reverse inclusion, under $P_1 \approx P_2$ we have: $(\nu\,\vec{y})P_1\{\vec{y}/\vec{z}\} \approx (\nu\,\vec{y})(P_1|\Pi[z_i \to y_i]^{\tau_i}) \approx (\nu\,\vec{y})(P_2|\Pi[z_i \to y_i]^{\tau_i}) \approx (\nu\,\vec{y})P_2\{\vec{y}/\vec{z}\}$, where the first and the last equations are by the copy-cat law (see Proposition 5.4), while the second equation is by closure of $\approx$ under parallel composition and hiding. Thus $P\mathbf{R}_3 Q$ implies $P \approx Q$, that is $\mathbf{R}_3 \subset \approx$. This shows $\mathbf{R}$ is indeed a bisimulation, hence as required.

### C.4. PROOF OF PROPOSITION 4.5

Let $\mathbf{R} \stackrel{\text{def}}{=} \{\langle P, Q \rangle \mid \mathfrak{I}_e \vdash P = Q\}$. The statement says $\mathbf{R} \subset \approx$. It is enough to show this inclusion under Convention 3.1 since $\approx$ is already closed under $\equiv$. By Proposition 4.3 (3), it suffices to show $\mathbf{R} \cup \equiv_\alpha$ is a bisimulation with respect to prime syntactic transition. We first consider the pair from (E1), $C[\overline{x}(\vec{y})P]|x(\vec{y}).Q$ and $C[(\nu\vec{y})(P|Q)]$. Let $R = C[\overline{x}(\vec{y})P]|x(\vec{y}).Q$ and set $\vdash R \triangleright A$. If $\vdash R \xrightarrow{l} R'$, we have the following cases.

$$\begin{array}{llll}
(1) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{l} C'[\overline{x}(\vec{y})P]|x(\vec{y}).Q & (2) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{\tau} (\nu\vec{y})(C[P]|Q) \\
(3) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{l} C[\overline{x}(\vec{y})P']|x(\vec{y}).Q & (4) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{\tau} C'[P]|x(\vec{y}).Q \\
(5) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{x(\vec{y})} C[\overline{x}(\vec{y})P]|Q & (6) & C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{\overline{x}(\vec{y})} C[P]|x(\vec{y}).Q.
\end{array}$$

We shall now show that only the processes in (1), (2) and (3) are typable. To this end we show by induction on the derivation of $\vdash R \triangleright A$ that $\updownarrow x \in A$. This implies that $C[\,]$ cannot contain an input at $x$. Hence (4) is not typable. Similarly, no typable observer could contain an output or an input at $x$, making in (5) and (6) untypable.

The transition (1) is matched by a transition $C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(P|Q)]$ while the empty transition sequence $C[(\nu\vec{y})(P|Q)]$ matches (2) because $C[(\nu\vec{y})(P|Q)] \equiv (\nu\vec{y})(C[P]|Q)$, as can be shown by induction on the derivation of (2). It is easy to see that $C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C[(\nu\vec{y})(P'|Q)]$ is an admissible match for (4).

Now assume that $R = C[(\nu\vec{y})(P|Q)] \xrightarrow{l} R'$, $\vdash R \triangleright A$ and $\vdash R \xrightarrow{l} R'$. We have the following causes of the transition.

$$\begin{array}{llll}
(1) & C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(P|Q)] & (2) & C[(\nu\vec{y})(P|Q)] \xrightarrow{\tau} C'[(\nu\vec{y})(\nu\vec{z})(P'|Q')] \\
(3) & C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(P'|Q)] & (4) & C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(P|Q')]
\end{array}$$

(1) is matched by $C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{l} C'[\overline{x}(\vec{y})P]|x(\vec{y}).Q$. For (2), we first show, by induction on the transition, that $C[\,]$ must be a reduction context. If the context $C[\,]$ in the definition of extended reduction (Definition 3.1) is restricted to a reduction context, then the resulting relation coincides with $\longrightarrow$, hence also with $\stackrel{\tau}{\Longrightarrow}$ by Proposition 4.2. Thus we obtain $C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{\tau} C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(\nu\vec{z})(P'|Q')]$ as matching transition sequence. The remaining cases (3) and (4) are dealt with in the same way.

Similarly for the pair from (E2), $C[\overline{x}(\vec{y})P]|!x(\vec{y}).Q$ and $C[(\nu\vec{y})(P|Q)]|!x(\vec{y}).Q$. Finally we can immediately reason about the pair from (E3), $(\nu x)!x(\vec{y}).P$ and $\mathbf{0}$, since no transition is possible in either process. ■

## C.5. PROOF OF LEMMA 4.1

For (1), if a $\triangleright$-normal form contains hiding and/or redundant $\mathbf{0}$, $\equiv'$ cannot equate it with the result of stripping them off; while if it doesn't, since $\triangleright$ only strips off (rather than increases) them, applying $\triangleright$ is the same thing as applying $\equiv'$. (2) is immediate by reaching an ENF by Theorem 3.2 and then by stripping redundant hiding and $\mathbf{0}$ by $\triangleright$ (which is inside $\equiv$). For (3), by definition the processes generated by the rules in Proposition 3.3 do not contain hiding and redundant $\mathbf{0}$. For the converse we argue by strucrural induction combined with these two conditions to show $\triangleright$-normal forms can be generated by the three rules in Proposition 3.3. (4) is immediate from (3). For (5), we show this for $P \in \mathsf{NF}_e$ which is enough. Suppose $P \in \mathsf{NF}_e$ and $P \xrightarrow{\tau} P'$. We can easily check $P \xrightarrow{\tau} P'$ implies $P \longrightarrow P'$, that is $P \mapsto P'$, which contradicts $P' \in \mathsf{NF}_e$. ■

# APPENDIX D

## PROOFS FOR SECTION 5

### D.1. PROOF OF PROPOSITION 5.2

For (1), let $\mathbf{R}$ be the result of adding an equation to $\cong$. By the definition of $\cong$ there are $P^{x:\mathbb{B}}\mathbf{R}Q^{x:\mathbb{B}}$ such that $P \Downarrow_x^1$ and $Q \Downarrow_x^2$. Take any $\vdash R_{1,2} \triangleright \uparrow ?A$ (with $x \notin \mathsf{fn}(A)$). Then $S \stackrel{\text{def}}{=} x[.R_1 \& .R_2]$ is typable. By the congruence of $\mathbf{R}$ we have $S|P_1\mathbf{R}S|P_2$. Since $\longrightarrow \in \cong$ (cf. Proposition 5.1) this implies $R_1\mathbf{R}R_2$. From such $R_{1,2}$ we can build any prime/non-prime terms, by which we conclude the universality of $\mathbf{R}$.

For (2-a), the "only if" direction is immediate from the definition. For the "if" direction, let $C[\cdot]$ be a context with its hole typed $A$ and the result typed $x:\mathbb{B}$ with $x$ fresh (if $x \in \mathsf{fn}(A)$ we can always use a copy-cat to mediate $x$ to a fresh name). Assume the latter condition and $C[P_1] \Downarrow_x^i$. If the hole of $C[\cdot]$ is not under an input prefix, then we already have $C[\ \cdot\ ] \stackrel{\text{def}}{=} (\nu\, \mathsf{fn}(A))(R|[\ \cdot\ ])$. Suppose the hole is under an input prefix. If $C[P_1] \Downarrow_x^i$ by $C[P_1] \twoheadrightarrow \overline{x}\mathtt{in}_i|C'[P_1\sigma]$ keeping $P_1$ under the input prefix along the way (possibly with some substitution $\sigma$) then we have $C[P_2] \twoheadrightarrow \overline{x}\mathtt{in}_i|C'[P_2]$, i.e. $C[P_2] \Downarrow_x^i$. If not, then suppose $C[P_1] \twoheadrightarrow C'[P_1\sigma]$ where $C'[P_1\sigma]$ is the first configuration in which the input prefix is taken off. Using copycats, we can represent $\sigma$ by parallel composition and hiding, so that the former condition gives us $C[P_2] \Downarrow_x^i$, as required.

(2-b) is immediate by performing extended reduction at occurrences of $x$ in $P$ on both sides, and noting $\mapsto \subset \approx \subset \cong$.

For (3), suppose $P \stackrel{\text{def}}{=} \overline{x}(\vec{y})P'$ has type $?A$ and take the context $C[\ ]$ from $A$ to $u:\mathbb{B}$. By (2) above we can set $C[\ ]$ has form $(\nu\, \vec{w})(S|R|[\ ])$ where $R$ is the composition of replicated processes compensating $A$. Since $u$ cannot occur in $R$ it occurs in $S$, whose behaviour at $u$ does not depend on $P$ in $C[P]$, i.e. $C[P] \Downarrow_u^i$ iff $C[\mathbf{0}] \Downarrow_u^i$, hence $P \cong \mathbf{0}$. (An alternative concise proof of (3) using a refined bisimulation is given in [71].) ■

### D.2. PROOF OF PROPOSITION 5.4

Both are mechanical by induction on $\tau$. Below we show the proof for (2), taking the unary replicated case. Let $\tau = (\vec{\rho})^!$ so that $[y \to x]^\tau \stackrel{\text{def}}{=} !y(\vec{z}).\overline{x}(\vec{w})\Pi[w_i \to z_i]^{\overline{\rho_i}}$. Let $P \equiv C[\overline{y}(\vec{z})R_1]..[\overline{y}(\vec{z})R_n]$ where $\overline{y}(\vec{z})R_j$ exhausts all prime outputs in $P$ (these contexts can be nested). Then we have:

$$\begin{array}{rcl}
(\nu\, y)(P|[y \to x]^\tau) & \mapsto^{n+1} & C[(\nu\, \vec{z})(R_1|\overline{x}(\vec{w})\Pi[w_i \to z_i]^{\overline{\rho_i}})]..[(\nu\, \vec{z})(R_n|\overline{x}(\vec{w})\Pi[w_i \to z_i]^{\overline{\rho_i}})] \\
& \equiv & C[\overline{x}(\vec{w})(\nu\, \vec{z})(R_1|\Pi[w_i \to z_i]^{\overline{\rho_i}})]..[\overline{x}(\vec{w})(\nu\, \vec{z})(R_n|\Pi[w_i \to z_i]^{\overline{\rho_i}})] \\
& \mapsto^* & C[\overline{x}(\vec{w})R_1\{\vec{z}/\vec{w}\}]...[\overline{x}(\vec{w})R_n\{\vec{z}/\vec{w}\}] \\
& \equiv_\alpha & C[\overline{x}(\vec{z})R_1]...[\overline{x}(\vec{z})R_n]
\end{array}$$

where: (1) the first extended reduction involves $n$ replications and 1 garbage collection; and (2) the second extended reduction in by induction hypothesis. This also gives the base case, where, with $\tau = ()^!$, we can dispense with the second steps forward. The final $\equiv_\alpha$ is possible because $\vec{z}$ are fresh w.r.t. $R_i$. Other cases are the same.

### D.3. PROOF OF PROPOSITION 5.6

We first show, by rule induction, $\vdash_\phi P^\sharp \triangleright A$ for some $\phi$ whenever $\vdash P \triangleright A$ for $P \in \mathsf{NF}_e$ and sequential $A$. W.l.o.g. we work under Convention 3.1. If $P = \mathbf{0}$ then $\vdash \mathbf{0} \triangleright ?A$ hence $\vdash_{\mathtt{I}} \mathbf{0} \triangleright A$, as required. For $P = P_1|P_2$, let $\vdash_{\phi_i} P_i \triangleright A_i$ $(i = 1,2)$ such that $A = A_1 \odot A_2$. Suppose $\phi_1 = \phi_2 = \circ$. Since $P \in \mathsf{NF}_e$, $P_1^\sharp$ and $P_2^\sharp$ respectively contains prime outputs $S_1^\sharp$ and $S_2^\sharp$ as factors of parallel composition. By $A$ being sequential, one of $S_1^\sharp$ and $S_2^\sharp$ has only ?-mode channels (note a prime output does not contain input subjects), which is impossible by construction. Thus one of $\phi_1$ and $\phi_2$ is $\imath$, from which sequential typability is immediate. The remaining cases are direct from the induction hypothesis. For $P \cong P^\sharp$, the only non-trivial case is $(\overline{x}(\vec{y})P)^\sharp \stackrel{\text{def}}{=} \mathbf{0}$ when $\uparrow \notin \mathsf{md}(A)$, for which we use Proposition 5.2 (3). The rest is direct from induction hypothesis.

## D.4. PROOF OF PROPOSITION 5.8

Let $T^\circ = (\vec{\tau})^!$ and $T_{1,2}$ arbitrary below. We first show:

(1) $\vdash \mathsf{Msg}\langle m,\vec{z}\rangle^T \triangleright m : \overline{T^\circ}, \vec{z} : \vec{\tau}$.
(2) $\vdash \mathsf{Arg}\langle m,N,\vec{z}\rangle^{T' \Rightarrow T} \triangleright m : \overline{(T' \Rightarrow T)^\circ}, \vec{z} : \vec{\tau}, E^\circ$ if $\vdash [\![N : T']\!]_u \triangleright u : T'^\circ \to E^\circ$.
(3) $\vdash \mathsf{Proj}_i\langle m,\vec{z}\rangle^{T_i} \triangleright m : \overline{(T_1 \times T_2)^\circ}, \vec{z} : \vec{\tau}$.
(4) $\vdash \mathsf{Sum}\langle m,\vec{z},\{(x_i)M_i\}\rangle^T \triangleright m : \overline{(T_1 + T_2)^\circ}, E^\circ$ if $\vdash [\![M_i : T_i']\!]_u \triangleright u : T_i'^\circ \to E^\circ$ $(i = 1,2)$ for some $T'_{1,2}$.

For proofs, (1) is immediate from Proposition 5.4 (1). The remaining statements are direct from the definition. For example, let $\vdash [\![N : T']\!]_u \triangleright u : T'^\circ, E^\circ$. For simplicity, assume $T^\circ = (\tau)^!$ and, accordingly, $\vec{z} = z$. By (1) we have, noting that $\mathsf{md}(\tau) = \uparrow$ in this case, $\vdash c(w).\mathsf{Msg}\langle w,z\rangle^{T^\circ} \triangleright c : (\overline{T^\circ})^\downarrow \to z : \tau$. Together with the given assumption, we obtain:

$$\vdash \overline{m}(nc)([\![N : T']\!]_n \mid c(w).\mathsf{Msg}\langle w,z\rangle^{T^\circ}) \triangleright m : (T'^\circ(\overline{T^\circ})^\downarrow)^?, z : \tau, E^\circ$$

as required. ■

*Remark.* In the above, $m$ in $\mathsf{Arg}\langle m,N,\vec{z}\rangle^{T' \Rightarrow T}$, $\mathsf{Proj}_i\langle m,\vec{z}\rangle^T$ and $\mathsf{Sum}\langle m,\vec{z},\{(x_i)M_i\}\rangle^T$ is typed by the dual of the encoded function, product and sum type, respectively, indicating the role of these expressions as the desctructors of the corresponding constructors.

Now we prove Proposition 5.8 by rule induction of the map $[\![M : T]\!]_u$.

**Case** $[\![x : T]\!]_u$**.** Direct from Proposition 5.4 (1).

**Case** $[\![() : \mathtt{unit}]\!]_u$**.** Immediate from the definition.

**Case** $[\![\lambda x : T'.M : T' \Rightarrow T]\!]_u$**.** Immediate from the induction hypothesis.

**Case** $[\![MN : T]\!]_u$**.** By induction hypothesis on $[\![N : T']\!]$ we can apply (2) above to $\mathsf{Arg}\langle m,N,\vec{z}\rangle^{T' \Rightarrow T}$). By induction hypothesis on $[\![E \vdash M : T' \Rightarrow T]\!]_m$ we have $\vdash [\![M : T' \Rightarrow T]\!]_m \triangleright m : (T' \Rightarrow T)^\circ \to E^\circ$. Thus, with $[\![T]\!] = (\vec{\tau})^!$:

$$\vdash (\nu\, mn)([\![M : T' \Rightarrow T]\!]_m \mid \mathsf{Arg}\langle m,N,\vec{z}\rangle^{T' \Rightarrow T}) \triangleright \vec{z} : \vec{\tau}, E^\circ$$

from which we obtain $\vdash !u(\vec{z}).(\nu\, mn)([\![M : T' \Rightarrow T]\!]_m \mid \mathsf{Arg}\langle m,N,\vec{z}\rangle^{T' \Rightarrow T}) \triangleright u : T^\circ \to E^\circ$.

**Case** $[\![\langle M_1, M_2\rangle : T_1 \times T_2]\!]_u$**.** Direct from the induction hypothesis.

**Case** $[\![\pi_i(M) : T]\!]_u$**.** By (3) above and induction hypothesis, arguing as in the case of $[\![MN : T]\!]_u$ in the last step.

**Case** $[\![\mathtt{inl}(M) : T + T']\!]_u$**.** Direct from the induction hypothesis.

**Case** $[\![\mathtt{case}\ L\ \mathtt{of}\ \{\mathtt{in}_i(x_i : T_i).M_i\}_{i \in \{1,2\}} : T]\!]_u$**.** By Lemma (2) above and by induction hypotheses, arguing as in the case of $[\![MN : T]\!]_u$ in the last step.

## D.5. PROOF OF PROPOSITION 5.9

The proof uses the following variation of the replication theorems [51, 55].

LEMMA D.1. $(\nu x)(C[[\![x:T]\!]_u]|[\![N:T]\!]_x) \mapsto^* (\nu x)(C[[\![N]\!]_u]|[\![N]\!]_x)$ assuming typability.

*Proof.* Let $T^\circ = (\vec{\tau})^!$ and $[\![N:T]\!]_x \stackrel{\text{def}}{=} !x(\vec{w}).P$. We have:

$$\begin{array}{rcl}(\nu x)(C[[\![x:T]\!]_u]|[\![N:T]\!]_x) & \stackrel{\text{def}}{=} & (\nu x)(C[!u(\vec{z}).\overline{x}(\vec{w})\Pi[w_i \to z_i]^{\tau_i}]|!x(\vec{w}).P) \\ & \mapsto & (\nu x)(C[!u(\vec{z}).(\nu\vec{w})(P|\Pi[w_i \to z_i]^{\tau_i})]|!x(\vec{w}).P) \\ & \mapsto^* & (\nu x)(C[!u(\vec{w}).P]|[\![N:T]\!]_x)\end{array}$$

The last step is by Proposition 5.4 (2). ■

We can now prove Proposition 5.9. For (β), assuming $[\![M]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).P$, we calculate:

$$\begin{array}{rll}[\![(\lambda x.M)N]\!]_u & \stackrel{\text{def}}{=} & !u(\vec{z}).(\nu m)(!m(xy).\overline{y}(m')[\![M]\!]_{m'}|\overline{m}(xy)([\![N]\!]_x|y(m').\mathsf{Msg}\langle m'\vec{z}\rangle)) \\ & \mapsto^2 & !u(\vec{z}).(\nu xm')([\![M]\!]_{m'}|[\![N]\!]_x|\mathsf{Msg}\langle m'\vec{z}\rangle) \\ & \mapsto^* & !u(\vec{z}).(\nu x)(P|[\![N]\!]_x) \qquad \text{(Prop. 5.4 (2))} \\ & \mapsto^* & !u(\vec{z}).P\{[\![N]\!]_{v_1}..[\![N]\!]_{v_n}/[\![x]\!]_{v_1}..[\![x]\!]_{v_n}\} \qquad \text{(Lemma D.1)} \\ & \stackrel{\text{def}}{=} & [\![M\{N/x\}]\!]_u,\end{array}$$

as required. For ($\text{proj}_1$), let $[\![M_1:T_1]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).P$ below.

$$\begin{array}{ll}[\![\pi_1\langle M_1,M_2\rangle:T_1]\!]_u & \\ \stackrel{\text{def}}{=} & !u(\vec{z}).(\nu m)(!m(c).\overline{c}(m_1m_2)\Pi[\![M:T_i]\!]_{m_i}|\overline{m}(c)c(m_1m_2).\mathsf{Msg}\langle m_1,\vec{z}\rangle) \\ \mapsto^3 & !u(\vec{z}).(\nu m_1m_2)(\Pi[\![M:T_i]\!]_{m_i}|\mathsf{Msg}\langle m_1,\vec{z}\rangle) \\ \mapsto^* & !u(\vec{z}).P \quad (\stackrel{\text{def}}{=} [\![M_1:T_1]\!]_u). \qquad \text{(Prop. 5.4 (2))}\end{array}$$

For ($\text{case}_i$), we again let $[\![M_1:T_1]\!]_u \stackrel{\text{def}}{=} !u(\vec{z}).P$.

$$\begin{array}{ll}[\![\mathtt{case}\ \mathtt{in}_1(N)\ \mathtt{of}\ \{\mathtt{in}_i(x_i:T_i).M_i\}:T]\!]_u & \\ \stackrel{\text{def}}{=} & !u(\vec{z}).(\nu l)(!l(c).\overline{c}\mathtt{in}_1(x_1)[\![N:T_1]\!]_{x_1}|\overline{m}(c)c[\&_i(x_i).(\nu m)([\![M_i:T]\!]_m|\mathsf{Msg}\langle m,\vec{z}\rangle)]) \\ \mapsto^3 & !u(\vec{z}).(\nu x_1)((\nu m)([\![M_i:T]\!]_m|\mathsf{Msg}\langle m,\vec{z}\rangle)|[\![N:T_1]\!]_{x_1}) \\ \mapsto^* & !u(\vec{z}).(\nu x_1)(P|[\![N:T_1]\!]_{x_1}) \qquad \text{(Prop. 5.4 (2))} \\ \mapsto^* & [\![M_1\{N/x_1\}:T]\!]_u, \qquad \text{(Lemma D.1)}\end{array}$$

hence done. ■

## D.6. PROOF OF LEMMA 5.3

We use one observation which is useful.

PROPOSITION D.1. *Let* $\vdash Q_{1,2} \triangleright A$ *and let* $\{\vec{y}\} \subset \mathsf{fn}(A)$ *such that* $\mathsf{md}(A(y_i)) = ?$ *for each* $y_i \in \{\vec{y}\}$*. Suppose for each* $\vdash R \triangleright \vec{y}:\overline{\vec{\tau}}$ *we have* $(\nu\vec{y})(Q_{1,2}|R) \Downarrow_e Q'$ *for some* $Q'$*. Then* $Q_1 \cong Q_2$*.*

In the statement above, observe we first choose an arbitrary ?-part of the given action type.

*Proof.* Below $Q_{1,2}$ and $A$ are as given above, and we let $B \stackrel{\text{def}}{=} \vec{y} : \vec{\tau}$ and $C \stackrel{\text{def}}{=} A/\vec{y}$.

$$\begin{array}{rcl} Q_1 \cong Q_2 & \Leftrightarrow & \forall \vdash R \triangleright \overline{A}, u{:}\mathbb{B}.\ (\nu\,\mathsf{fn}(A))(Q_1|R) \Downarrow_x^i \ \Leftrightarrow\ (\nu\,\mathsf{fn}(A))(Q_2|R) \Downarrow_x^i) \\ & \Leftrightarrow & \forall \vdash S \triangleright \overline{B}. \forall \vdash T \triangleright \overline{C}, u{:}\mathbb{B}.\ (\nu\,\mathsf{fn}(A))(Q_1|S|T) \Downarrow_x^i \ \Leftrightarrow\ (\nu\,\mathsf{fn}(A))(Q_2|S|T) \Downarrow_x^i) \\ & \Leftrightarrow & \forall \vdash S \triangleright \overline{B}.\ (\nu\,\mathsf{fn}(B))(Q_1|S) \cong (\nu\,\mathsf{fn}(B))(Q_2|S) \\ & \Leftarrow & \forall \vdash S \triangleright \overline{B}.\ (\nu\,\mathsf{fn}(B))(Q_1|S) \approx (\nu\,\mathsf{fn}(B))(Q_2|S) \\ & \Leftarrow & \forall \vdash S \triangleright \overline{B}.\ \exists Q'.\ (\nu\,\mathsf{fn}(B))(Q_1|S) \Downarrow_e Q' \text{ and } (\nu\,\mathsf{fn}(B))(Q_2|S) \Downarrow_e Q'. \end{array}$$

The first equivalence is by Proposition 5.2 (2). The second equivalence is by taking each compensating replicated term to be a prime whose only free name is its subject (this does not lose generality since, by extended reduction with other replicated processes, all ?-moded free names can be compensated and eliminated). For the third equivalence, the "only if" direction is by contradiction, while the "if" direction is by noting, at type $x{:}\mathbb{B}$, the correspondence in convergence and $\cong$ coincide. The last two (reverses) implications are, respectively, by $\approx \subset \cong$ (cf. Proposition 5.3) and by $\leftrightarrow \subset \approx$ (cf. Theorem 4.1). ■

Below we only show the case for $\mathtt{let}\ x = zF\ \mathtt{in}\ F'$ (other cases are easier). Write $(\ )^\star_u$ for $[\![(\ )^\circ]\!]_u$. Assume $[\![F']\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P$. Then we have, using the induction hypothesis, extended reductions and Lemma 5.2 (2-b),

$$\begin{array}{rcl} (\mathtt{let}\ x = zF\ \mathtt{in}\ F')^\star_u & \stackrel{\text{def}}{=} & [\![F'^\circ\{zF^\circ/x\}]\!]_u \ \approx\ (\nu x)(F'^\star_u | (zF)^\star_x) \\ & \cong & (\nu x)(!u(\vec{v}).P \mid [\![zF]\!]_x) \\ & \cong & (\nu x f)(!u(\vec{v}).P \mid !x(\vec{w}).\mathsf{Arg}\langle z f \vec{w}\rangle \mid [\![F]\!]_f) \end{array}$$

Let the resulting term be $S_1$. We compare $S_1$ with the direct translation $S_2 \stackrel{\text{def}}{=} !u(\vec{v}).\overline{z}(fc)([\![F']\!]_f | c(x).P)$. For this purpose we compose $S_i$ with $Q \stackrel{\text{def}}{=} !z(fc)\overline{c}(x)!x(\vec{w})Q'$. Note that we can assume any compensating process at $z$ has this form without loss of generality (by $\eta$-expansion). Immediately $S_1|Q \mapsto^+ (\nu x f)(!u(\vec{v}).P \mid !x(\vec{w}).Q' \mid [\![F]\!]_f) \mid Q$, while, for the right-hand side, we have:

$$S_2|Q \ \mapsto^+\ !u(\vec{v}).(\nu f x)([\![F']\!]_f | !x(\vec{w})Q'|P) \mid Q \ \cong\ (\nu f x)(!u(\vec{v}).P \mid !x(\vec{w}).Q' \mid [\![F']\!]_f).$$

We can now use Proposition D.1. ■

## D.7. PROOF OF LEMMA 5.4

For (1), write $Q_1$ (resp. $Q_2$) for the process on the l.h.s. (resp. r.h.s.) of the equation for brevity. By the conditions on typability and by ENFs, we should have $\vdash_\circ Q_{1,2} \triangleright z{:}\tau^\uparrow, x{:}\rho^!, ?A$ for some $A$. We prove the following claim, which easily entails (cf. Proposition D.1 in D.6) the required equality.

**Claim.** *Assume, for each* $\vdash_{\mathtt{I}} U \triangleright \overline{A}, x{:}\overline{\rho}$, *we have* $(\nu\,\mathsf{fn}(A)x)(Q_i|U) \Downarrow_e Q'$ $(i = 1,2)$ *for some* $Q'$. *Then* $Q_1 \cong Q_2$.

To prove the claim, let $A = \emptyset$ for simplicity, which does not lose generality since processes compensating $A$ are simply absorbed into $Q_{1,2}$ by extended reduction, resulting processes in the same shape. For the same reason we safely assume the occurrence of $x$ in each term is unique. Thus we compose $\vdash_{\mathtt{I}} !x(\vec{r}c).S \triangleright x{:}\overline{\rho}$ to both sides and demonstrate the claim. By

Proposition 3.3 and by noting $\mathsf{fn}(R|S) \subset \{\vec{r}c\}$, we have $(\nu\,\vec{r})(R|S) \Downarrow_e \overline{c}\mathtt{in}_i(\vec{w}_i)T$ for some $T$. Assume further $(\nu\,\vec{w}_i)(T|P_i) \Downarrow_e P_i'$. For $Q_1$ we obtain:

$$\begin{array}{rcl}(\nu\,x)(Q_1 \mid !x(\vec{r}c).S) & \stackrel{\text{def}}{=} & (\nu\,x)(\overline{x}(\vec{r}c)(R \mid c[\&_i(\vec{w}_i).\overline{z}(e)!e(\vec{y}).P_i]) \mid !x(\vec{r}c).S) \\ & \mapsto^+ & (\nu\,\vec{r}c)(R \mid S \mid c[\&_i(\vec{w}_i).\overline{z}(e)!e(\vec{y}).P_i]) \\ & \mapsto^* & (\nu\,c)(\overline{c}\mathtt{in}_i(\vec{w}_i)T \mid c[\&_i(\vec{w}_i).\overline{z}(e)!e(\vec{y}).P_i]) \\ & \mapsto & (\nu\,\vec{w}_i)(T \mid \overline{z}(e)!e(\vec{y}).P_i) \\ & \mapsto^* & \overline{z}(e)!e(\vec{y}).P_i'\end{array}$$

For $Q_2$ we have:

$$\begin{array}{rcl}(\nu\,x)(Q_2 \mid !x(\vec{r}c).S) & \stackrel{\text{def}}{=} & (\nu\,x)(\overline{z}(e)!e(\vec{y}).\overline{x}(\vec{r}c)(R \mid c[\&_i(\vec{w}_i).P_i]) \mid !x(\vec{r}c).S) \\ & \mapsto^+ & \overline{z}(e)!e(\vec{y}).(\nu\,\vec{r}c)(R \mid S \mid c[\&_i(\vec{w}_i).P_i]) \\ & \mapsto^* & \overline{z}(e)!e(\vec{y}).(\nu\,\vec{w}_i)(T \mid P_i) \\ & \mapsto^* & \overline{z}(e)!e(\vec{y}).P_i'\end{array}$$

hence as required.

For (2), since a linear output cannot occur freely under a replicated input, given a sequential $!u(\vec{x}z).P$ with $z$ of type $(\tau)^{\uparrow}$, we can write $P$ as $C[\overline{z}(c)P_1]..[\overline{z}(c)P_n]$ where either $P \equiv \overline{z}(c)P'$ or each hole is under a linear unary/branching prefix. Treating w.l.o.g. unary prefix as a special case of branching prefix, we prove (2) by induction on the depth of $z$ in $P$, where the *depth of $z$ in $P$* is the maximum number of prefixes from the subject of $P$ to an occurrence of $z$ in $P$. If the depth is zero, there is nothing to prove. Let the depth be $n+1$ and let $P \equiv C[\overline{x}(\vec{r}e)(R|e[\&_i.\overline{z}(c)!c(\vec{y}).P_i])]$ where the mentioned occurrence of $x$ is the deepest one (by which they can only occur immediately after $e$). By (1) above, $P \cong C[\overline{z}(c)!c(\vec{y}).\overline{x}(\vec{r}e)(R|e[\&_i.P_i])]$. By induction hypothesis we are done.

(3) is because the transformation in the proof of (2) can be carried out incrementally. ■