Strong Normalisation in the π-Calculus *

Nobuko Yoshida † Martin Berger ⋆ Kohei Honda ⋆

Abstract

We introduce a typed π-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 λ-calculi and linear logic with process-theoretic reasoning. It is adaptable to systems involving state 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 λ-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.

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 logical specification of program behaviour. One important property in this context, widely found in typed λ-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 λ-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 so far.

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. Strong normalisation 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) [17, 38] on the basis of a simply typed λ-calculus. Such languages 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: as 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 the first-order strongly normalising λ-calculi are fully abstractly embeddable. The type discipline simply adds the minimum form of causal chains to the system introduced in [9] where we established a fully abstract encoding of PCF. This small addition radi-

*A short version will appear in *Proc. LICS'01*, the 16th Annual IEEE Symposium on Logic and Computer Science, Boston, Massachusetts, June 16-19, 2001.

†Department of Mathematics and Computer Science, University of Leicester, University Road, Leicester, LE1 7RH, UK. E-Mail: `nyl1@mcs.le.ac.uk`. ⋆Department of Computer Science, Queen Mary, University of London, London, E1 4NS, UK. E-Mail: {`martinb, kohei`}`@dcs.qmw.ac.uk`. Partially supported by EPSRC grant GR/N/37633.

cally 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 [7, 15, 42], combined with process-algebraic reasoning [9, 35, 37, 41, 47]. 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 extensions of the presented formalism. In the following, we outline key technical ideas and relate our work to the existing literature.

The π-Calculus Following [9], we use an asynchronous variant of the π-calculus [10, 21] as our base syntax. 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. 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 at a to have returned a value via a private name c

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

where $\overline{a}(c)\, c(y).P$ stands for $(\nu c)(\overline{a}\langle c\rangle \,|\, c(y).P)$ with (νc) being a restriction operator. Using these agents, R_a below is a simple but interesting example of deadlock

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

since R_a causes an infinite reduction sequence and the receptor $c(y).P$ waits forever for an answer at c. In an untyped setting, R_a is equal to $\overline{a}(c)\, c(y).P$ up to asynchronous bisimilarity, but the two are quite different regarding resource consumption. The next example shows how subtleties arise through new link creation.

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

After a one step reduction via a, we obtain $\mathtt{Fw}\langle bc\rangle \,|\, \mathtt{Fw}\langle cb\rangle \,|\, \overline{b}$ which exhibits divergence.

Type discipline for strong normalisability The type discipline of this paper is a simple refinement of [9]. Concretely, the system is based on two central ideas:

- *Linear types* [14, 30, 31, 47], 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 [9, 28, 34, 37, 41].
- *Action types with causality*, where causality is represented by edges in a directed graph whose acyclicity ensures the absence of circular dependencies [30, 31, 47]. Transmission of causality is controlled by a form of *cut elimination* in action types.

Let us illustrate these points by examples. First, $\mathtt{Fw}\langle ab\rangle$ is typed as follows, assuming an appropriate environment Γ.

$$\Gamma \vdash \mathtt{Fw}\langle ab\rangle \triangleright\, !a \rightarrow ?b$$

Here $!a \rightarrow ?b$ indicates that the process repeatedly inputs at a and then outputs at b. Cut elimination occurs between input and output with the same name. For example, given an appropriate base Γ, we can type (⊗ being disjoint union):

$$\Gamma \vdash \mathtt{Fw}\langle a_1c\rangle \,|\, \mathtt{Fw}\langle a_2c\rangle \,|\, \mathtt{Fw}\langle cb\rangle \triangleright\, !a_1{\rightarrow}?b \otimes !a_2{\rightarrow}?b \otimes !c{\rightarrow}?b$$

$$\Gamma \vdash\, !a(x).(\overline{b}\langle x\rangle \,|\, \overline{b}\langle x\rangle) \,|\, !b(x).(\overline{c}\langle x\rangle \,|\, \overline{c}\langle x\rangle) \triangleright (!a \rightarrow ?c) \otimes (!b{\rightarrow}?c)$$

We can detect a cyclic dependency such as $\mathtt{Fw}\langle ab\rangle \,|\, \mathtt{Fw}\langle ba\rangle$ by looking at their types $!a \rightarrow ?b$ and $!b \rightarrow ?a$ [22, 28, 47].

Proving SN for the π-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, e.g. the reductions of $\overline{a}\langle v\rangle|\mathtt{Fw}\langle ab\rangle|\mathtt{Fw}\langle bc\rangle$ proceed at a, b and c in this order, but in $\overline{a}\langle v\rangle|\mathtt{Fw}\langle ab\rangle|\mathtt{Fw}\langle ba\rangle$ are repeated between a and b. However, due to creation of new links and replication of terms, both crucial features of π-calculi, such reasoning is infeasible, at least in its naive form. Consider the process

$$!a(x).(\overline{x}\langle v_1\rangle|\overline{x}\langle v_2\rangle) \,|\, \overline{a}(c)\mathtt{Fw}\langle cb\rangle \,|\, !b(x).(\overline{a}\langle x\rangle|\overline{a}\langle x\rangle) \qquad (1)$$

which has type $!a \otimes !b$. 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 λ-term $\lambda xyz.((xz)(yz))$ [33]. The difficulty in analysing (1) can be seen by considering the following subterm of a one step descendant of (1).

$$(\nu\, c)(\overline{c}\langle v_1\rangle \,|\, \overline{c}\langle v_2\rangle \,|\, \mathtt{Fw}\langle cb\rangle)$$

It contains a chain $!c \rightarrow ?b$, which is difficult to determine before c is passed. In fact, if we naively represent causality incorporating bound names in (1), 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 λ-calculi [7, 12].

The idea we use is suggested by SN proofs for typed λ-calculi, due to, among others, Tait [42]. His method uses a semantic interpretation of each type $[\![\sigma]\!]$ as a collection of strongly normalising λ-terms, and shows that all typable terms are indeed in these sets. A key step is to prove, for each $M : \tau$ (for which by induction $M \in [\![\tau]\!]$), that $\lambda x : \sigma.M \in [\![\sigma \to \tau]\!]$, 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 ρ, mapping each variable of type σ to some term in $[\![\sigma]\!]$. Now the result is immediate. While we cannot use an identical framework due to the different nature of reduction in the π-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_i which complements P. Termination behaviour is calculated via the extended reduction suggested by strong bisimilarity (which does not change termination) together with replication theorems [9, 35, 41]. Then acyclicity in causality yields strong normalisation.

Summary of Contributions The following summarises main technical contributions of the present work. (3) solves an open problem in [33] for the simple type hierarchy.

1. Introduction of a typed π-calculus where strong normalisability is ensured by typability. SN has significant consequences for the calculus, including the finite axiomatisation of the weak bisimilarity and the basic liveness in interaction.
2. Establishment of strong normalisability of typable processes combining ideas from traditional SN proofs for typed λ-calculi with process-theoretic reasoning.
3. Embedding, using Milner's encoding [33], of the simply typed λ-calculus with sums and products ($\lambda_{\to,\times,+}$) into our typed π-calculus. The embedding is fully abstract w.r.t. the observational congruence of $\lambda_{\to,\times,+}$, justifying all commutative conversions and η-equations [15] and automatically leads to SN in the source calculus.

Related Work Strong normalisation in typed λ-calculi has been studied extensively in the past; detailed surveys can be found in [7, 12]. Abramsky extends the Curry-Howard correspondence to linear logic [14] using proof expressions 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. The operational structure of [5] follows his own π-calculus encoding of proof nets [2]. The appeal of realisability lies in treating semantics and syntax uniformly on a logical basis. In the context of SN types for the π-calculus, sharing of names and dynamic link creation would make the framework in [1, 5] hard to apply directly. In contrast, the present work offers a possibly basic type discipline that does not directly correspond to known logical systems but is based on simple operational principles, resulting in a new effective method to ensure SN for name passing processes.

As our initial example of server-client interaction suggests, SN in processes is closely related to liveness properties in interaction. Yoshida [47] presents a typed π-calculus with a local liveness property. Kobayashi and his colleagues [26, 28–30] propose several typing systems which ensure a form of liveness (in [29] time quotas are assigned to communications for this purpose). Unlike the present work, these and other preceding typing systems for π-calculus [9, 11, 18, 19, 39, 41] do not guarantee SN and the associated liveness properties for processes involving non-trivial use of replication. As a result, embeddability of, say, $\lambda_{\to}$ in these systems does not guarantee the SN of the source calculus in these systems.

Structure of the Paper Section 2 introduces the syntax and the type discipline. Section 3 proves the main result, the strong normalisability. Section 4 presents the complete axiomatisation of weak bisimilarity. Section 5 gives a fully abstract encoding of $\lambda_{\to,\times,+}$. Section 6 briefly outlines further results. The appendix lists omitted proofs.

2. Processes and Typing

2.1. Processes

Following [9], we use the asynchronous version of the π-calculus [10, 21] with bound output [40]. [1]

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

The bound/free names are defined as usual and we assume the variable convention for bound names. The structural rules are standard except for omission of $!P \equiv !P|P$ and

[1] The full syntax includes branching, which is discussed in Section 5.2.

for incorporation of congruence rules making output asynchronous.

$$\begin{array}{lcll} \overline{x}(\vec{z})\,(P\,|\,Q) & \equiv & (\overline{x}(\vec{z})P)\,|\,Q & \text{if } \mathsf{fn}(Q)\cap\{\vec{z}\}=\emptyset \\ \overline{x}(\vec{z})(\nu w)P & \equiv & (\nu w)\overline{x}(\vec{z})P & \text{if } w\notin\{x\vec{z}\} \end{array}$$

The reduction $\longrightarrow$ is generated by the following rules.

$$\begin{array}{lcl} x(\vec{y}).P\,|\,\overline{x}(\vec{y})\,Q & \longrightarrow & (\nu\vec{y})(P\,|\,Q) \\ !x(\vec{y}).P\,|\,\overline{x}(\vec{y})\,Q & \longrightarrow & !x(\vec{y}).P\,|\,(\nu\vec{y})(P\,|\,Q) \end{array}$$

The relation is defined over processes modulo $\equiv$ and is closed under parallel composition, restriction and output.

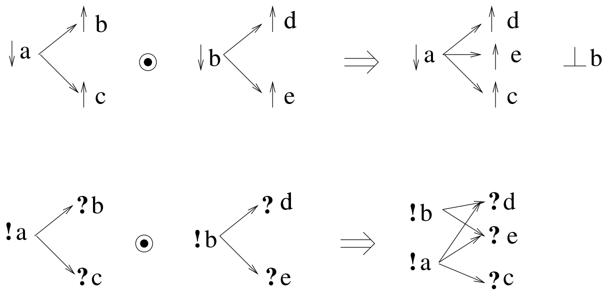

Figure 1. Composition between Action Types

2.2. Types

Channel Types The following pairs of *action modes* [9, 22] prescribe how each channel is used in typed processes.

$\downarrow$	Linear input	$\uparrow$	Linear output
!	Replicated input	?	Output to !

We also use $\perp$ to indicate the presence of both input and output at a linear channel. $p,q,\ldots$ range over action modes. For $p\neq\perp$, 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 modes correspond to $!_1, ?_1, !_\omega$ and $?_\omega$ introduced in [9], except that the present modes indicate true linearity for linear channels (i.e. input and output occurs precisely once) and lack of divergence for replicated channels.

Using action modes, we first define the set of *channel types*: they are assigned to names and indicate how channels would be used.

$$\begin{array}{lcl@{\qquad}lcl} \alpha & ::= & \langle\tau,\,\overline{\tau}\rangle & \tau_{\mathtt{I}} & ::= & (\vec{\tau}_{\mathtt{O}})^{\downarrow} \;\;|\;\; (\vec{\tau}_{\mathtt{O}})^{!} \\ \tau & ::= & \tau_{\mathtt{I}} \;|\; \tau_{\mathtt{O}} & \tau_{\mathtt{O}} & ::= & (\vec{\tau}_{\mathtt{I}})^{\uparrow} \;\;|\;\; (\vec{\tau}_{\mathtt{I}})^{?} \end{array}$$

In the first line $\overline{\tau}$ denotes the *dual* of τ, which is the result of dualising all action modes; $\mathsf{md}(\tau)$ indicates the (outermost) action mode of τ. A type of form $\langle\tau,\overline{\tau}\rangle$, called a *pair type*, is an unordered pair of mutually dual types.

Following [9] we only consider types where, in $(\vec{\tau}_{\mathtt{O}})^{\downarrow}$, each τ_i has mode ? (and dually for $(\vec{\tau}_{\mathtt{I}})^{\uparrow}$). This constraint, which comes from game semantics, is not essential for SN but simplifies presentation and proofs.

Action Types 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 [47] and witness the real usage of channels. Formally, an *action type*, denoted $A,B,\ldots$, is a finite directed graph with nodes of the form px, such that:

- no names occur twice; and
- edges are of the form $!x\to ?y$ or $\downarrow x\to\uparrow y$.

If px is in A and for no y we have $qy\to px$, 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. $A\backslash\vec{x}$ is the result of taking off nodes with names in $\vec{x}$ from A. $A\uplus B$ is the graph union of A and B.

Now define a symmetric partial operator $\odot$ by: $\downarrow\odot\uparrow=\perp$, $?\odot ?=?$ and $!\odot ?=!$. Write $A\asymp B$ iff:

- whenever $px\in A$ and $qx\in B$, $p\odot q$ is defined; and
- whenever $p_1x_1\to\overline{p}_2x_2,\, p_2x_2\to\overline{p}_3x_3,\ldots,\; p_nx_n\to\overline{p}_{n+1}x_{n+1}$ in $A\uplus B$ ($n\geq 1$), we have $x_1\neq x_n$.

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

- $px\in|A\odot B|$ iff either (1) $px\in|A|$ and $x\notin\mathsf{fn}(B)$ and the symmetric case, or (2) $qx\in|A|$ and $rx\in|B|$ and $p=q\odot r$.
- $px\to qy$ in $A\odot B$ iff both (1) $px,qy\in|A\odot B|$ and (2) $px=r_1z_1\to\overline{r}_2z_2,\, r_2z_2\to\overline{r}_3z_3,\ldots,r_nz_n\to\overline{r}_{n+1}z_{n+1}=qy$ in $A\uplus B$ ($n\geq 1$).

We can easily check that $\odot$ is a symmetric and associative partial operation on action types with unit $\emptyset$.

Example 2.1 Figure 1 shows examples of composition between action types. In the linear case, ordering from/to node b disappears. On the other hand, in the replicated input case, we need to keep the original ordering because $!b(\vec{y}).P$ remains persistently. We can write down these examples syntactically as follows (shared ?-nodes are duplicated in syntax): $\downarrow a\to(\uparrow b\otimes\uparrow c)\odot\downarrow b\to(\uparrow d\otimes\uparrow e)=\downarrow a\to(\uparrow c\otimes\uparrow d\otimes\uparrow e)\otimes\perp b$, and $!a\to(?b\otimes ?c)\odot !b\to(?d\otimes ?e) = !a\to(?c\otimes ?d\otimes ?e)\otimes !b\to(?d\otimes ?e)$.

2.3. Linear Typing

We are now ready to present the typing rules. Sequents have the form $\Gamma\vdash P\triangleright A$ where Γ is a finite map from names to

$$\text{(Zero)}\ \frac{-}{\Gamma \vdash \mathbf{0} \rhd \emptyset} \qquad \text{(Par)}\ \frac{\Gamma \vdash P_i \rhd A_i \quad (i=1,2) \qquad A_1 \asymp A_2}{\Gamma \vdash P_1 \,|\, P_2 \rhd A_1 \odot A_2} \qquad \text{(Res)}\ \frac{\Gamma \cdot x:\alpha \vdash P \rhd A \qquad px \in |A| \text{ with } p \in \{\bot, !\}}{\Gamma \vdash (\nu x:\alpha)P \rhd A/x}$$

$$\text{(Weak-}\bot\text{)}\ \frac{\Gamma \vdash x : \downarrow,\uparrow \qquad \Gamma \vdash P \rhd A^{-x}}{\Gamma \vdash P \rhd A \otimes \bot x} \qquad \text{(Weak-?)}\ \frac{\Gamma \vdash x : ? \qquad \Gamma \vdash P \rhd A^{-x}}{\Gamma \vdash P \rhd A \otimes ?x}$$

$$\text{(In}^{\downarrow}\text{)}\ (C/\vec{y} = \uparrow A \otimes ?B)\ \frac{\Gamma \vdash x : (\vec{\tau})^{\downarrow} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \rhd C^{-x}}{\Gamma \vdash x(\vec{y}:\vec{\tau}).P \rhd (\downarrow x \to A) \otimes B} \qquad \text{(Out}^{\uparrow}\text{)}\ (C/\vec{y} = A \asymp \uparrow x)\ \frac{\Gamma \vdash x : (\vec{\tau})^{\uparrow} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \rhd C}{\Gamma \vdash \overline{x}(\vec{y}:\vec{\tau})P \rhd A \odot \uparrow x}$$

$$\text{(In}^{!}\text{)}\ (C/\vec{y} = ?A)\ \frac{\Gamma \vdash x : (\vec{\tau})^{!} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \rhd C^{-x}}{\Gamma \vdash !x(\vec{y}:\vec{\tau}).P \rhd !x \to A} \qquad \text{(Out}^{?}\text{)}\ (C/\vec{y} = A \asymp ?x)\ \frac{\Gamma \vdash x : (\vec{\tau})^{?} \qquad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \rhd C}{\Gamma \vdash \overline{x}(\vec{y}:\vec{\tau})P \rhd A \odot ?x}$$

Figure 2. Linear Typing Rules

channel types, called a *base*. The typing rules are given in Figure 2. The following notation is used.

$A/\vec{x}$	$A\backslash\vec{x}$ such that $x_i \in \mathsf{active}(A)$ for each x_i
pA	A such that $\mathsf{md}(A) = \{p\}$
A^{-x}	A such that $x \notin \mathsf{fn}(A)$

Further, $px \to A$ adds new edges from px to active nodes in A, $A \otimes B$ (resp. $\Gamma \cdot \Delta$) denotes the disjoint union of A and B (resp. Γ and Δ), and $\Gamma \vdash x : \tau$ denotes either $x : \tau$ or $x : \langle \tau, \overline{\tau} \rangle$ is in Γ. The sequent $\Gamma \vdash P \rhd A$ is often abbreviated to $\Gamma \vdash P$.

We briefly illustrate the typing rules. In (Par), "$\asymp$" controls composability, ensuring linearity of channels, and prevents circular dependency. In (Res), we do not allow $\uparrow$, ? or $\downarrow$-channels to be restricted since they expect their dual actions to exist in the environment (cf. [9, 19, 22, 30]). In addition to recording causality, (In$^{\downarrow}$) ensures that x occurs precisely once (by C^{-x}) and that no free input is suppressed under the prefix. (Out$^{\uparrow}$) also ensures that x occurs precisely once but permits suppression by the prefix since output is asynchronous. (In$^{!}$) is the same as (In$^{\downarrow}$) except that no free $\uparrow$-channels are suppressed (if a $\uparrow$-channel is under replication then it can be used more than once). (Out$^{?}$) and (Weak-?) say that ?-channels occur zero or more times and do not suppress actions.

Example 2.2

- A *copy-cat* copies all information from one channel to another [4, 25]. We show, step by step, how $[u \to x]^{\tau} \stackrel{\text{def}}{=} !u(a).\overline{x}(b)b.\overline{a}$, the copy-cat from u to x, can be typed. Let $\tau = ((\,)^{\uparrow})^{!}$, $\Gamma = a : (\,)^{\uparrow} \cdot b : (\,)^{\downarrow} \cdot u : \tau \cdot x : \overline{\tau}$. Then: (1) $\Gamma \vdash \mathbf{0} \rhd \emptyset$, (2) $\Gamma \vdash \overline{a} \rhd \uparrow a$, (3) $\Gamma \vdash b.\overline{a} \rhd \downarrow b \to \uparrow a$, (4) $\Gamma\backslash b \vdash \overline{x}(b)b.\overline{a} \rhd ?x \otimes \uparrow a$ (by $(\downarrow b \to \uparrow a)/b = \uparrow a$) and finally (5) $\Gamma\backslash ab \vdash !u(a).\overline{x}(b)b.\overline{a} \rhd !u \to ?x$ (by $(?x \otimes \uparrow a)/a = ?x$).
- Let $\alpha = \langle (\,)^{\downarrow}, (\,)^{\uparrow} \rangle$ and $\Gamma = a{:}\alpha \cdot b{:}\alpha \cdot c{:}\alpha \cdot d{:}\alpha$. Then $\Gamma \vdash a.(\overline{b} \,|\, \overline{c}) \rhd \downarrow a \to (\uparrow b \otimes \uparrow c)$ and $\Gamma \vdash b.\overline{d} \rhd \downarrow b \to \uparrow d$. By (Par), $\Gamma \vdash a.(\overline{b} \,|\, \overline{c}) \,|\, b.\overline{d} \rhd \downarrow a \to (\uparrow c \otimes \uparrow d) \otimes \bot b$, and by (Res), $\Gamma \vdash (\nu b)(a.(\overline{b} \,|\, \overline{c}) \,|\, b.\overline{d}) \rhd \downarrow a \to (\uparrow c \otimes \uparrow d)$.
- Let $\Gamma = x{:}\langle \tau, \overline{\tau} \rangle \cdot y{:}\langle \tau, \overline{\tau} \rangle \cdot z{:}\langle \tau, \overline{\tau} \rangle$ and $\tau = ((\,)^{\uparrow})^{!}$. Then the connection of two links (copy-cats) is typed as:

$$\Gamma \vdash [x \to y]^{\tau} \,|\, [y \to z]^{\tau} \rhd (!x \to ?z) \otimes (!y \to ?z)$$

with $(!x \to ?y) \odot (!y \to ?z) = (!x \to ?z) \otimes (!y \to ?z)$. However, $[x \to x]^{\tau}$ and $[x \to y]^{\tau} \,|\, [y \to x]^{\tau}$ are untypable under any environment by the side condition C^{-x} in (In$^{!}$) and by definition of $\asymp$, respectively.

Next we list two properties of name usage in typed processes. Acyclicity becomes crucial in our SN proof later.

Proposition 2.3 *Let $\Gamma \vdash P \rhd A$.*

i. (linearity) *If $px \in A$ such that $p \in \{\downarrow, \uparrow, !\}$, then x occurs precisely once in P.*
ii. (acyclicity) *$G(P)$ denotes a directed graph s.t.:* (1) *nodes are $\mathsf{fn}(P)$; and* (2) *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 occurs free in 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: (i) is mechanical. For (ii), we show, if $\Gamma \vdash P \rhd A$ then $px \to qy$ in A iff $x \curvearrowright \ldots \curvearrowright y$ is a maximal non-cycle in P, which is proved simulteneously with: if there are name-disjoint $p_1x_1 \to p_2x_2$, $p_3x_3 \to p_4x_4$, $\ldots$, $p_{2n-1}x_{2n-1} \to p_{2n}x_{2n}$ then the corresponding maximal non-cycles do not overlap in names, by rule induction on typing rules. The key case is (Par) which is the only rule which extends the chain. Assume $p_1x_1 \to \overline{p}_2x_2, p_2x_2 \to p_3x_3, \ldots p_{n-1}x_{n-1} \to \overline{p}_nx_n$ in $A \uplus B$ and $p_1x_1 \to \overline{p}_nx_n$ in $A \odot B$. By (IH) there are the corresponding maximal non-cycles. In them, names used in different cycles in A (resp. in B) never overlap with each other. Further, since intermediate names in these non-cycles have either mode ! or mode $\bot$, 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 $p_1x_1 \to \overline{p}_nx_n$, as required. ■

Remark. In Proposition 2.3 (ii), the notion of a chain does not include the cases when an intermediate channel is restricted. While such cases can be included, they are not necessary in the proof of strong normalisability given later, cf. Lemma 3.8.

Some notation which we use later:

- $P \Downarrow Q \stackrel{\text{def}}{\Leftrightarrow} P \longrightarrow^* Q \not\longrightarrow$.
- $P \Downarrow \stackrel{\text{def}}{\Leftrightarrow} \exists Q.P \Downarrow Q$. 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 2.4 *Let* $\Gamma \vdash P \triangleright A$.

i. (subject reduction) *If* $P \longrightarrow^* Q$ *then* $\Gamma \vdash Q \triangleright A$.
ii. (one-step confluence) *If* $P \longrightarrow Q_i$ $(i = 1,2)$ *with* $Q_1 \not\equiv Q_2$ *then there exists* R *s.t.* $Q_i \longrightarrow R$ $(i = 1,2)$.
iii. (determinacy) (1) $P \longrightarrow P'$ *and* $\mathsf{SN}(P')$ *imply* $\mathsf{SN}(P)$. (2) $P \Downarrow Q_i$ $(i = 1,2)$ *imply* $Q_1 \equiv Q_2$. *And* (3) $P \Downarrow \Leftrightarrow \mathsf{SN}(P) \Leftrightarrow \mathsf{CSN}(P)$.

PROOF: (i) is as in [9]. For (ii), we note that the only critical pairs arise when a replicated input is shared which does not change its shape. (iii) is standard, cf. [1, 6, 24, 27], all using (ii). See Appendix A. ■

3. Strong Normalisation

This section proves the following result.

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

A few significant consequences of the theorem will be discussed in Sections 4, 5 and 6. 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* $[\![\Gamma, A]\!]$, 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 show all normal forms to be in their semantic types. Then we establish that each typable process combined with resources always reaches a normal form, which implies the strong normalisability of $\longrightarrow$. In the second stage acyclicity (cf. Proposition 2.3) becomes crucial.

3.1. Extended Reduction

Definition 3.2 (extended reductions) We define $\mapsto_{\mathsf{l}}$, $\mapsto_{\mathsf{r}}$ and $\mapsto_{\mathsf{g}}$ as the compatible relations on processes modulo $\equiv$ respectively generated by the following rules.

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

Here $C[\,]$ is an arbitrary context not binding x. Then $\mapsto \stackrel{\text{def}}{=} (\mapsto_{\mathsf{l}} \cup \mapsto_{\mathsf{r}} \cup \mapsto_{\mathsf{g}})$ is the *extended reduction* relation.

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 β/linear law [18, 19, 30, 47], the replication law [9, 37, 41] and the garbage collection law, respectively. 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 $\mathsf{CSN}(P)$, using $\mapsto$ instead of $\longrightarrow$. A $\mapsto$*-redex* is a pair of terms which form a redex for $\mapsto$ in a given term. We say process P is *prime with subject* x if either P is input with subject at x or $P \equiv \overline{x}(y_1..y_n)\Pi_i P_i$ such that each P_i is prime with subject y_i. We assume all prefixed terms to be primes throughout the rest of the section (this loses no generality, see Appendix B.3). NF_e is given by $\{\Gamma \vdash P, P \not\mapsto\}$. Note that a process is in NF_e if it does not contain complementary input and output and, moreover, it does not have substantial hiding (i.e. a hiding $(\nu\,x)P$ such that $x \in \mathsf{fn}(P)$). Thus we can see NF_e is inductively generated by the following rules up to $\equiv$ (implicitly assuming typability):

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

Proposition 3.3 *Let all processes be typed below.*

i. *If* $\Gamma \vdash P \triangleright A$ *and* $P \mapsto P'$ *then* $\Gamma \vdash P' \triangleright A$.
ii. (CR) *If* $P \mapsto^* Q_i$ *then* $Q_i \mapsto^* R$ $(i = 1,2)$.
iii. (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* $\mathsf{CSN}_e(P)$.

Note that the Church-Rosser property is no longer one-step. The proof proceeds by 'postponing' applications of $\mapsto_{\mathsf{g}}$. See Appendix B.1.

3.2. Semantic Types

Semantic types are provably strongly normalising typed terms of some kind. In the present first-order system, it suffices to define them inductively as interpretations of syntactic types. We need some preliminaries.

$$
\begin{array}{rcll}
[\![\Gamma,\ A]\!] & \stackrel{\text{def}}{=} & \begin{cases} \{\Gamma \vdash P \triangleright A \mid P \Downarrow_e \mathbf{0}\} & (A = \emptyset) \\ \{\Gamma \vdash P \triangleright A \mid \forall Q \in [\![\overline{\Gamma},\ \mathsf{cl}(A)]\!].\ P|Q \Downarrow_e \Pi_{i\in I} R_i \ \text{ with } R_i \in \langle\!\langle \Gamma \cup \overline{\Gamma},\ p_i x_i \rangle\!\rangle\} & (A \cdot \mathsf{cl}(A) = \otimes_{i\in I} p_i x_i) \end{cases} \\
\langle\!\langle \Gamma,\ \downarrow x \rangle\!\rangle & \stackrel{\text{def}}{=} & \{x(\vec{y} : \vec{\tau}).P \mid P \in [\![\Gamma \cdot \vec{y} : \vec{\tau},\ \otimes p_i y_i]\!] \ \text{ with } \Gamma \vdash x : (\vec{\tau})^{\downarrow} \text{ and } p_i = \mathsf{md}(\tau_i)\} \\
\langle\!\langle \Gamma,\ !x \rangle\!\rangle & \stackrel{\text{def}}{=} & \{!x(\vec{y} : \vec{\tau}).P \mid P \in [\![\Gamma \cdot \vec{y} : \vec{\tau},\ \otimes p_i y_i]\!] \ \text{ with } \Gamma \vdash x : (\vec{\tau})^{!} \text{ and } p_i = \mathsf{md}(\tau_i)\}
\end{array}
$$

Figure 3. Prime and Non-Prime Semantic Types

- $\mathsf{cl}(A) \stackrel{\text{def}}{=} \otimes_{p_i x_i \in A, p_i \in \{\uparrow, ?\}} \overline{p_i} x_i$.
- Let $A \asymp B$ and $A \odot B = C \otimes \perp \vec{x}$ where $\perp \notin \mathsf{md}(C)$. Then $A \cdot B \stackrel{\text{def}}{=} C$.

By $\mathsf{cl}(A)$, called the *closure of* A, we indicate the (type of the) environment which, when composed with A, give complementary linear and replicated inputs for the free output channels in A. $A \cdot B$ is a "semantic version" of $A \odot B$, where we forget inessential $\perp$-channels. We write $\Gamma \vdash A$ if modes in A conform to Γ. We can now define semantic types.

Definition 3.4 The *semantic type* $[\![\Gamma,\ A]\!]$ of a pair Γ and A such that $\Gamma \vdash A$, and the *prime semantic type* $\langle\!\langle \Gamma,\ px \rangle\!\rangle$ for a pair Γ and px such that $\Gamma \vdash px$ with $p \in \{\downarrow, !\}$, are defined by the rules in Figure 3.

In the second line of Figure 3, $\overline{\Gamma}$ and $\Gamma \cup \overline{\Gamma}$ denote the result of dualising all types in Γ and the name-wise union of types, respectively. From the definition, we can always assume Γ in $[\![\Gamma,\ A]\!]$ is *paired*, i.e. contains only pair types, with no loss of generality. The rules are well-defined since the height of types decreases in effect (see Appendix B.2 for detail). Some observations:

Lemma 3.5

i. *If* $P \in [\![\Gamma,\ A]\!]$ *then* $\Gamma \vdash P \triangleright A$ *and* $\mathsf{SN}_e(P)$.

ii. $[\![\Gamma,\ A]\!] \subseteq [\![\Gamma,\ A \otimes B]\!]$. *Also* $[\![\Gamma,\ A \otimes \perp x]\!] = [\![\Gamma,\ A]\!]$.

iii. *Let* $P \mapsto P'$. *Then* $P \in [\![\Gamma,\ A]\!]$ *iff* $P' \in [\![\Gamma,\ A]\!]$.

iv. *Let* $P_i \in \langle\!\langle \Gamma,\ p_i x_i \rangle\!\rangle$ $(1 \le i \le n)$ *such that* $x_1, .., x_n$ *are pairwise distinct. Then* $\Pi_i P_i \in [\![\Gamma,\ \otimes_i p_i x_i]\!]$.

PROOF: For (i), the first half is immediate, while the latter is because $P|Q \Downarrow_e$ implies $P \Downarrow_e$ and $Q \Downarrow_e$. The first half of (ii) is direct from definition. The latter is by: $\Gamma \vdash P \triangleright A \otimes \perp x$ with $x \in \mathsf{fn}(P)$ implies there exists P' s.t. $P \mapsto_{\mathsf{l}} P'$ with $\Gamma/x \vdash P \triangleright A$. For (iii), "then" is trivial, while "if" is by $\mapsto$ being CR. Finally, (iv) is because $\mathsf{cl}(\otimes p_i x_i) = \emptyset$ in this case. ■

3.3. Main Proofs

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 augument possible.

Lemma 3.6 *If* $\Gamma \vdash P \triangleright A$ *and* $P \in \mathsf{NF}_e$ *then* $P \in [\![\Gamma,\ A]\!]$.

PROOF: By Lemma 3.5 (ii), it suffices to consider only minimum action types. 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 induction on the structure of P. We only list two cases, see Appendix B.4 for the remaining cases.

(Inaction). By $\mathsf{cl}(\emptyset) = \emptyset$ and $\mathbf{0}|Q \equiv Q \Downarrow_e \mathbf{0}$ for each $Q \in [\![\Gamma,\ \emptyset]\!]$, immediately $\mathbf{0} \in [\![\Gamma,\ \emptyset]\!]$.

(Replicated Output). Assume $P \in [\![\Gamma \cdot \vec{y} : \vec{\tau},\ C \otimes ?x]\!]$ with $C/\vec{y} = \uparrow A \otimes ?B^{-x}$ (cf. Appendix B.3). We have to show $\overline{x}(\vec{y} : \vec{\tau})P \in [\![\Gamma,\ A \otimes B \otimes ?x]\!]$. First we note that $\mathsf{cl}(A \otimes B \otimes ?x) = \mathsf{cl}(C \otimes ?x) = (\overline{A} \otimes \overline{B} \otimes !x)$. Assume $Q \in [\![\Gamma, \overline{A} \otimes \overline{B} \otimes !x]\!]$. W.l.o.g. we can write $Q \equiv !x(\vec{y}).Q_0' \,|\, Q_1 \,|\, Q_2$ where $!x(\vec{y}).Q_0' \in \langle\!\langle \Gamma,\ !x \rangle\!\rangle$, $Q_1 \equiv \Pi_i Q_{1i\langle \downarrow a_i \rangle}$ 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 (IH), $P|!x(\vec{y}).Q_0'|Q_1|Q_2 \Downarrow_e P'|!x(\vec{y}).Q_0'|Q_2$ s.t. $P' \in [\![\Gamma \cdot \vec{y} : \vec{\tau},\ \otimes p_i y_i]\!]$ with $p_i = \mathsf{md}(\tau_i) \in \{!, \downarrow\}$. 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 $Q_0' \in [\![\Gamma \cdot \vec{y} : \vec{\overline{\tau}},\ \otimes \overline{p}_i y_i]\!]$. Hence, by assumption,

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

Now by CR, we have $P \,|\, Q \Downarrow_e !x(\vec{y}).Q_0' \,|\, Q_2 \in [\![\Gamma,\ \overline{B} \otimes !x]\!]$, as desired. ■

Corollary 3.7 *If* $\Gamma \vdash P \triangleright px \in \mathsf{NF}_e$ *with* $p \in \{\downarrow, !\}$, *then* $P \in \langle\!\langle \Gamma,\ px \rangle\!\rangle$.

We can now establish the main lemma below: given the Lemma 3.6, Prefix and restriction become trivial, given Lemma 3.6, but parallel composition causes problems. Even if $!a.\overline{b}$ and $(\overline{a}\,|\,!b.\overline{c})$ are in 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:

$$\begin{array}{rlcl} & !a.\overline{b}\,|\,\overline{a}\,|\,!b.\overline{c}\,|\,!c.\mathbf{0} & \mapsto_{\mathsf{r}} & !a.\overline{b}\,|\,\overline{a}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0} \\ \mapsto_{\mathsf{r}} & !a.\mathbf{0}\,|\,\overline{a}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0} & \mapsto_{\mathsf{r}} & !a.\mathbf{0}\,|\,!b.\mathbf{0}\,|\,!c.\mathbf{0} \end{array}$$

This reduction strategy always terminates due to acyclicity of names. Formally, we prove:

Lemma 3.8 (main lemma) *Suppose* $\Gamma \vdash P \triangleright A$. *Then* $P|Q \Downarrow_e$ *for each* $Q \in [\![\overline{\Gamma},\ \mathsf{cl}(A)]\!]$.

PROOF: By induction on the typing rules. (Zero) and (Weak-$\bot$,-?) are trivial. For the prefix rules, by induction the body of each prefix converges, hence so does the whole term. Then we use Lemma 3.6 again. (Res) is similar, by Lemma 3.5 (ii). For (Par), suppose $\Gamma \vdash P_i \triangleright A_i$ 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:

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

Since $i \searrow^+ j \searrow^+ i$ implies the existence of a cycle $x \curvearrowright^+ x$ in the sense of Proposition 2.3 (ii), $\searrow^*$ is a partial order on X. We now define a series of sets $X_0, X_1, ..$ as follows, writing $\mathsf{max}(Y, \leq)$ for the set of maximal elements of a partially ordered set Y.

$$X_1 \stackrel{\text{def}}{=} \mathsf{max}(X, \searrow^*) \qquad X_{i+1} \stackrel{\text{def}}{=} \mathsf{max}(X \backslash \textstyle\bigcup_{0 \leq j \leq i} X_i, \searrow^*)$$

As X is finite, the $X_1, .., X_m$ partition X for some m. Now let $S_i \stackrel{\text{def}}{=} \Pi_{j \in X_i} Q_j$ for $0 \leq i \leq m$. By construction $P \equiv \Pi_{1 \leq i \leq m} S_i$ and $S_i \in \mathsf{NF}_e$ for each i. Note that 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 $\Gamma \vdash S_i \triangleright C_i$ for each i so that $\odot_{1 \leq i \leq m} C_i = A$ 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 $E_1 = \mathsf{cl}(C_1) = \mathsf{cl}(A)$ and $E_m = \mathsf{cl}(A) \odot (A)$. Choose any $R \in [\![\Gamma,\ \mathsf{cl}(A)]\!]$. We now show by induction on $1 \leq l \leq m+1$ that for some $R_l \in [\![\Gamma,\ E_l]\!]$

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

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 $R_l \in [\![\Gamma,\ E_l]\!]$. By Lemma 3.6 and by $S_l \in \mathsf{NF}_e$ we know that $S_{l_1} \in [\![\Gamma,\ C_{l+1}]\!]$. With $E_l = \mathsf{cl}(C_1) \odot C_1 \odot .. \odot C_{l-1}$, this implies that we have $S_l|R_l \Downarrow_e R' \in [\![\Gamma,\ E_{l+1}]\!]$. We can now set $R' \equiv R_{l+1}$, as desired. ■

Theorem 3.9 (strong normalisability in $\mapsto$) $\Gamma \vdash P \triangleright A$ *implies* $\mathsf{CSN}_e(P)$.

By $\longrightarrow \subseteq \mapsto$ and Proposition 2.4 (iii-3), we have now established Theorem 3.1.

4. Characterisation of Bisimilarity

As a striking consequence of the strong normalisability of typed processes, this section shows that weak bisimilarity has a finite axiomatisation.

4.1. Typed Transitions and Typed Bisimulations

Typed transitions describe the observations a typed observer can make of a typed process. Typed transitions are a proper subset of untyped transitions while not restricting τ-actions: hence typed transitions restricts observability, not computation. First, *untyped transitions* $P \stackrel{l}{\longrightarrow} Q$, with labels τ, $x(\vec{y})$ and $\overline{x}(\vec{y})$ are generated by the following rules.

$$x(\vec{y}).P \xrightarrow{x(\vec{y})} P \quad \overline{x}(\vec{z})P \xrightarrow{\overline{x}(\vec{z})} P \quad !x(\vec{y}).P \xrightarrow{x(\vec{y})} P|!x(\vec{y}).P$$

The communication and contextual rules are standard except for closure under asynchronous output.

$$P \stackrel{l}{\longrightarrow} P' \text{ with } \mathsf{fn}(l) \cap \{\vec{y}\} = \emptyset \quad \Rightarrow \quad \overline{x}(\vec{y})\,P \stackrel{l}{\longrightarrow} \overline{x}(\vec{y})\,P'$$

Typed transitions, written $\Gamma \vdash P \stackrel{l}{\longrightarrow} \Gamma \cdot \vec{y}{:}\vec{\tau} \vdash Q$, where $\vec{y}{:}\vec{\tau}$ assigns names introduced in l as prescribed by Γ, are generated as follows, cf. Section 4.2 and Appendix E of [9]: $\Gamma \vdash P \stackrel{l}{\longrightarrow} \Gamma \cdot \vec{y}{:}\vec{\tau} \vdash Q$ iff (1) $\Gamma \vdash P \triangleright A$, (2) $P \stackrel{l}{\longrightarrow} Q$ with $\mathsf{bn}(l) \cap \mathsf{fn}(\Gamma) = \emptyset$, (3) if $\bot x \in |A|$ then $\mathsf{fn}(l) \neq x$, and (4) if $!x \in |A|$ and $\mathsf{active}(l) = x$ then l is input.

Using typed transitions, we define bisimulations. Let us say a relation over typed processes is *typed* if it relates only processes with identical base and action type. A typed relation is a *typed congruence* when it is a typed equivalence closed under typed contexts, contains $\equiv$ and allows weakening of bases in the standard way [9, 37]. A typed relation $\mathbf{R}$ is a *weak bisimulation*, or a *bisimulation*, if $\Gamma \vdash P\mathbf{R}Q$ implies: whenever $\Gamma \vdash P \stackrel{l}{\longrightarrow} P'$ then there is a typed transition sequence $\Gamma \vdash Q \stackrel{\hat{l}}{\Longrightarrow} Q'$ such that $P\mathbf{R}Q$, as well as the symmetric case. By replacing $\stackrel{\hat{l}}{\Longrightarrow}$ with $\stackrel{l}{\longrightarrow}$, we obtain a *strong bisimulation*. If $\Gamma \vdash P\mathbf{R}Q$ for some weak (resp. strong) bisimulation $\mathbf{R}$, we write $\Gamma \vdash P \approx Q$ (resp. $\Gamma \vdash P \sim Q$). Finally, $\approx$ (resp. $\sim$) is called *weak* (resp. *strong*) *bisimilarity*. The weak bisimilarity is often simply called *bisimilarity*.

4.2. Characterisation

Let $\longleftrightarrow$ be the transitive, symmetric closure of $\mapsto \cup \equiv$. We shall now show that $\longleftrightarrow$ completely characterises bisimilarity. For details see Appendix C.

Definition 4.1

- The relation $\equiv'$ is the least congruence such that $\equiv_\alpha \subseteq \equiv'$, $P|Q \equiv' Q|P$ and $(P|Q)|R \equiv' P|(Q|R)$.
- The relation $\rhd$ is the least typed precongruence containing $\equiv'$ such that $P|\mathbf{0} \rhd P$, $(\nu x)\mathbf{0} \rhd \mathbf{0}$, $P|(\nu x)Q \rhd (\nu x)(P|Q)$ if $x \notin \mathsf{fn}(P)$, $P|\overline{x}(\vec{y})Q \rhd \overline{x}(\vec{y})(P|Q)$ if $\mathsf{fn}(P) \cap \{\vec{y}\} = \emptyset$ and $\overline{x}(\vec{y})(\nu z)P \rhd (\nu z)\overline{x}(\vec{y})P$ if $z \notin \{x, \vec{y}\}$.
- P is in $\rhd$-*normal form* if $P \in \mathsf{NF}_e$ and $P \rhd Q$ implies $P \equiv' Q$.

Clearly $\rhd$-normal forms are representatives of NF_e, in fact precisely those generated by the rules in §3.1.

Lemma 4.2 *If $\Gamma \vdash P \rhd A$ then there is a $\rhd$-normal form Q such that $P \mapsto^* Q$.*

The proof of Lemma 4.2 is by construction together with Theorem 3.9.

Lemma 4.3 *Let P and Q be two typable $\rhd$-normal forms. Then $P \approx Q$ iff $P \equiv' Q$ iff $P \longleftrightarrow Q$.*

The key observation for the proof of Lemma 4.3 is that $\rhd$-normal forms are a class of processes where trace equivalence, $\approx$ and $\equiv'$ (hence also $\sim$ and $\equiv$) coincide.

Theorem 4.4 *(characterisation of $\approx$)* (i) $\mapsto\, \subsetneq\, \approx$, $\mapsto^* \subsetneq \approx$; *and* (ii) $\longleftrightarrow\, = \,\approx$.

The proof for (i) essentially proceeds by showing $\mathbf{R} \cup \mathbf{id}$ to be a typed bisimulation where $\mathbf{R}$ is inductively generated by the following rules:

$$\begin{array}{rcl} C[\overline{x}(\vec{y})P]|x(\vec{y}).Q & \mathbf{R} & C[(\nu\,\vec{y})(P|Q)] \\ C[\overline{x}(\vec{y})P]|!x(\vec{y}).Q & \mathbf{R} & C[(\nu\,\vec{y})(P|Q)]\,|\,!x(\vec{y}).Q \\ (\nu x)\,!x(\vec{y}).Q & \mathbf{R} & \mathbf{0} \end{array}$$

Here $C[\,]$ is an arbitrary context not binding x.

To establish (ii), assume that $P \approx Q$. By Lemma 4.2 we can find $\rhd$-normal forms P_{nf} and Q_{nf} of P and Q respectively such that $P \mapsto^* P_{nf}$ and $Q \mapsto^* Q_{nf}$. Hence by (i) $P_{nf} \approx Q_{nf}$. But Lemma 4.3 implies that $\approx$ restricted to $\rhd$-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 λ-calculus with products and sums ($\lambda_{\rightarrow,\times,+}$ from now on) as a testbed for the expressiveness of the presented 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 $\{1,2\}$.

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

We write $M \equiv_\alpha N$ for the α-equality on terms. A term is *closed* if no variables occur free.

The typing rules are standard (cf. [16, 36]). 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 is written $\rightsquigarrow$ (see Appendix D for details). We take the minimum meaningful reduction for simplicity, but the main technical results in this section hold for all reasonable variations. We write $M \Downarrow N$ when $M \rightsquigarrow^* N$ and $N \not\rightsquigarrow$. A *normal form* is a term which has no further reductions. By easy induction on the structure of terms, a closed normal form of type $T \rightarrow T'$ (resp. $T \times T'$, $T + T'$) has shape $\lambda x.M$ (resp. $\langle M,N \rangle$, $\mathtt{in}_i(M)$).

Equality in $\lambda_{\rightarrow,\times,+}$ is not as simple as it may look, due to the existence of sums [14]. To have a semantically meaningful equality, we use observation of "values", cf.[36]. Let $\mathtt{true} \stackrel{\mathrm{def}}{=} \mathtt{in}_1(())$ and $\mathtt{false} \stackrel{\mathrm{def}}{=} \mathtt{in}_2(())$, both of type $\mathbb{B} \stackrel{\mathrm{def}}{=} \mathtt{unit} + \mathtt{unit}$. Then $E \vdash M \cong_\lambda N : T$ when, for each context $C[\,]_T : \mathbb{B}$ such that $C[M]$ and $C[N]$ are closed, we have $(C[M] \Downarrow \mathtt{true} \Leftrightarrow C[N] \Downarrow \mathtt{true})$. The same equality is obtained by taking observability at each sum type, justifying all commuting conversions and η-rules.

5.2. The π-Calculus: Extension with Branching

Before the encoding, we extend the typed π-calculus to its full syntax [9] by incorporating branching. Branching is necessary to represent sums in $\lambda_{\rightarrow,\times,+}$ and is also used for defining a reduction-based typed congruence [23, 47].

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

The additional rules for $\longrightarrow$, $\mapsto$ and linear typing are given in Appendix D (for easy reference Appendix F offers the full syntax and related definitions about the calculus at a glance). All arguments and results in the preceding sections carry over to the full syntax without alteration.[2]

Let us say A is *closed* when $\mathsf{md}(A) \subseteq \{!, \perp\}$. Now write $\Gamma \vdash P \Downarrow^i_x$ when $P \Downarrow (\nu\,\vec{y})(\overline{x}\mathtt{in}_i(\vec{z})P_0|R)$ with $x \notin \{\vec{y}\}$ where

[2] A minor change is Proposition 2.3 (i): for a $\uparrow$-channel, "precisely once" becomes, under a branching input, "precisely once in each branch".

$$
\begin{array}{llll}
\textbf{(Type)} & \mathtt{unit}^\circ \overset{\mathrm{def}}{=} ((\,)^\uparrow)^! & (T_1 \Rightarrow T_2)^\circ \overset{\mathrm{def}}{=} (\overline{T_1^\circ}(T_2^\circ)^\uparrow)^! & (T_1 \times T_2)^\circ \overset{\mathrm{def}}{=} ((T_1^\circ T_2^\circ)^\uparrow)^! \qquad (T_1 + T_2)^\circ \overset{\mathrm{def}}{=} ([T_1^\circ \otimes T_2^\circ]^\uparrow)^! \\
\textbf{(Base)} & \emptyset^\circ \overset{\mathrm{def}}{=} \emptyset & (E \cdot x : T)^\circ \overset{\mathrm{def}}{=} E^\circ \cdot x : \overline{T^\circ} &
\end{array}
$$

(Terms) (if $T_2 = T_{12} \Rightarrow T_{22}$ then $\vec{z} = z_1 z_2$ else $\vec{z} = z$)

$$\llbracket x : T \rrbracket_u \overset{\mathrm{def}}{=} [u \to x]^{T^\circ}$$

$$\llbracket MN : T_2 \rrbracket_u \overset{\mathrm{def}}{=} !u(\vec{z}).(\nu mn)(\llbracket M : T_1 \Rightarrow T_2 \rrbracket_m \mid \llbracket N : T_1 \rrbracket_n \mid \mathsf{Arg}\langle mn\vec{z}\rangle^{T_1 \Rightarrow T_2})$$

$$\llbracket \lambda x : T_1 . M : T_1 \Rightarrow T_2 \rrbracket_u \overset{\mathrm{def}}{=} !u(xz).\overline{z}(m)\llbracket M : T_2 \rrbracket_m$$

$$\llbracket \langle M_1, M_2 \rangle : T_1 \times T_2 \rrbracket_u \overset{\mathrm{def}}{=} !u(c).\overline{c}(m_1 m_2)(\llbracket M_1 : T_1 \rrbracket_{m_1} \mid \llbracket M_2 : T_2 \rrbracket_{m_2})$$

$$\llbracket \pi_1\, M : T_1 \rrbracket_u \overset{\mathrm{def}}{=} !u(\vec{z}).(\nu m)(\llbracket M : T_1 \times T_2 \rrbracket_m \mid \mathsf{Proj}_1\langle m\vec{z}\rangle^{T_1})$$

$$\llbracket (\,) : \mathtt{unit} \rrbracket_u \overset{\mathrm{def}}{=} !u(x).\overline{x}$$

$$\llbracket \mathtt{inl}(M) : T_1 + T_2 \rrbracket_u \overset{\mathrm{def}}{=} !u(c).\overline{c}\mathtt{inl}(m)\llbracket M : T_1 \rrbracket_m$$

$$\llbracket \mathtt{case}\ L\ \mathtt{of}\ \mathtt{inl}(x_1)M_1\ \mathtt{or}\ \mathtt{inr}(x_2)M_2 : T \rrbracket_u \overset{\mathrm{def}}{=} !u(\vec{z}).(\nu l)(\llbracket L : T_1 + T_2 \rrbracket_l \mid \mathsf{Sum}\langle l\vec{z}, (x_i)M_i\rangle^T)$$

$$\mathsf{Arg}\langle mn\vec{z}\rangle^{T_1 \Rightarrow T_2} \overset{\mathrm{def}}{=} \overline{m}(n'c')([n' \to n]^{T_1^\circ} \mid c'(u').\mathsf{Con}\langle u'\vec{z}\rangle^{T_2^\circ})$$

$$\mathsf{Proj}_i\langle m\vec{z}\rangle^T \overset{\mathrm{def}}{=} \overline{m}(e)e(v_1 v_2).\mathsf{Con}\langle v_i \vec{z}\rangle^{T^\circ}$$

$$\mathsf{Sum}\langle l\vec{z}, (x_i)M_i\rangle^T \overset{\mathrm{def}}{=} \overline{l}(c)c[\&_{1,2}(x_i).(\nu m)(\llbracket M_i : T \rrbracket_m \mid \mathsf{Con}\langle m\vec{z}\rangle^{T^\circ})]$$

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

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

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

We omit $\mathtt{inr}(M)$ and $\pi_2 M$. For the copy-cats for unary types we assume the indexing set to be singletons.

Figure 4. Encoding of $\lambda_{\to,\times,+}$

$\Gamma \vdash P \triangleright A \otimes \uparrow x$ with A closed. We then define $\cong_{\mathsf{sn}}$ as the maximum typed congruence such that if $\Gamma \vdash P \cong_{\mathsf{sn}} Q$ and $\Gamma \vdash P \Downarrow_x^i$ then $\Gamma \vdash Q \Downarrow_x^i$ (cf. [23, 47]). We use the following two lemmas about $\cong_{\mathsf{sn}}$.

Lemma 5.1 (context lemma) *Let $\Gamma \vdash P_j \triangleright A$ ($j = 1,2$) with Γ paired. Then $P_1 \cong_{\mathsf{sn}} P_2$ iff: $P_1|R \Downarrow_x^i \;\Leftrightarrow\; P_2|R \Downarrow_x^i$ for each $\Gamma \cdot x : [\oplus_{1,2}]^\uparrow \vdash R \triangleright B$ s.t. $A \asymp B$.*

Lemma 5.2 (permutation) *Let $\overline{x}\mathtt{in}_l(\vec{y})P$ be prime, w have mode ? under Γ and $\{\vec{z}\} \cap \mathsf{fn}(\overline{x}\mathtt{in}_l(\vec{y})P) = \emptyset$. Then, assuming typability,*

$$\begin{array}{ll}\Gamma \vdash & \overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).(\overline{x}\mathtt{in}_l(\vec{y})P|R_i)]) \cong_{\mathsf{sn}} \\ & \overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).R_i]) \mid \overline{x}\mathtt{in}_l(\vec{y})P\end{array}$$

Similarly when x or b or both are unary.

The first lemma is proved following [9]. The proof of the second lemma uses Lemma 5.1 and CR of $\mapsto$. See D.3 and D.4 for details.

5.3. Embedding and Full Abstraction

The encoding of $\lambda_{\to,\times,+}$ is given in Figure 5. It adapts Milner's call-by-name encoding [33] to our type structure by adding an indirection at each λ-abstraction. The basic correspondence result follows. Note that in the second statement, there is an exact operational correspondence between $\rightsquigarrow$ and $\mapsto$: $\rightsquigarrow$ is simulated by $\mapsto$ directly, not up to some semantic equality.

Proposition 5.3 *Let $E \vdash M : T$ below with $\mathsf{fn}(E) = \{\vec{y}\}$.*

i. *$E^\circ \cdot u : T^\circ \vdash \llbracket M : T \rrbracket_u \triangleright !u \to ?\vec{y}$ is well-typed.*

ii. *$E \vdash M \rightsquigarrow M' \;\Rightarrow\; \llbracket M \rrbracket_u \mapsto^+ \llbracket M' \rrbracket_u$.*

Corollary 5.4 *$\lambda_{\to,\times,+}$ is strongly normalising.*

PROOF: By Theorem 3.9 using: if $\llbracket N_1 \rrbracket_u \equiv \llbracket N_2 \rrbracket_u$ with N_i in normal form then $N_1 \equiv_\alpha N_2$. ■

The above corollary offers a faithful *computational embedding* of $\lambda_{\to,\times,+}$: we now show that this also extends to semantics. First, by Proposition 5.3 and Corollary 5.4:

Lemma 5.5 (computational adequacy) *Let $M : \mathbb{B}$ be closed. Then $M \Downarrow \mathtt{true}$ iff $\llbracket M \rrbracket_u \Downarrow_e \llbracket \mathtt{true} \rrbracket_u$.*

Corollary 5.6 (soundness) $\llbracket E \vdash M : T \rrbracket_u \cong_{\mathsf{sn}} \llbracket E \vdash N : T \rrbracket_u$ *implies $E \vdash M \cong_\lambda N : T$.*

For completeness, we use a specific class of linear processes. Let us say P is *sequential* [9] if it is typable by the same system as Figures 2 and D.2 augmented with the sequentiality constraint in Figure 1 in [9]. See Appendix D for details. A key lemma for completeness follows.

Lemma 5.7 (sequential testability) *Let $E \overset{def}{=} \vec{y}{:}\vec{S}$ and $E^\circ \cdot u : T^\circ \vdash P_n \triangleright !u \otimes ?\vec{y}$ ($n = 1,2$). Then $P_1 \cong_{\mathsf{sn}} P_2$ iff:*

$$(\Pi_j R_j \mid P_1 \mid Q) \Downarrow_x^i \quad \Leftrightarrow \quad (\Pi_j R_j \mid P_2 \mid Q) \Downarrow_x^i \qquad (i = 1,2)$$

for each sequential $y_j : S_j^\circ \vdash R_j \triangleright\, !y_j$ *and sequential* $u : \overline{T^\circ}, x : [\oplus_{1,2}]^\uparrow \vdash Q \triangleright ?u \otimes \uparrow x$.

PROOF: By Lemma 5.1 and by assuming, via Theorem 3.9, the context to be in NF_e, we obtain $\Pi_j R_j$ and Q of desired types. We then transform each R_j and Q to their sequential variants, using Lemma 5.2. See Appendix D.7. ■

The final step is to show that each process with $\lambda_{\rightarrow,\times,+}$-types is translatable to a *canonical normal form* [4, 25] (CNF) whose grammar is given below.

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

We omit the typing and reduction rules. CNFs are translated to $\lambda_{\rightarrow,\times,+}$-terms in the standard way without changing meaning up to $\cong_\lambda$. Thus we have an encoding of CNFs into the π-calculus, also written $[\![F : \alpha]\!]_u$ which satisfies Lemma 5.5 (a direct encoding of CNFs in the π-calculus is also given in Appendix D.8). We can now prove:

Proposition 5.8 (definability) *Let* $E^\circ \cdot u : T^\circ \vdash P \triangleright\, !u \rightarrow ?\vec{y}$ *sequential with* $\mathsf{fn}(E) = \{\vec{y}\}$. *Then* $P \cong_{\mathsf{sn}} [\![F]\!]_u$ *for some* $E \vdash F : T$

The proof is by induction on the size of sequential processes under all $\lambda_{\rightarrow,\times,+}$-bases and types, using permutations (Lemma 5.2). For details, see Appendix D.9. We can now establish full abstraction.

Theorem 5.9 (full abstraction) $E \vdash M_1 \cong_\lambda M_2 : T$ *iff* $E^\circ \cdot u : T^\circ \vdash [\![M_1 : T]\!]_u \cong_{\mathsf{sn}} [\![M_2 : T]\!]_u$.

PROOF: By Corollary 5.6 (soundness), we only have to show the "then" direction. Suppose $M_1 \cong_\lambda M_2$ but $[\![M_1]\!]_u \not\cong_{\mathsf{sn}} [\![M_2]\!]_u$. By the latter, take ΠR_j and Q as in Lemma 5.7 s.t. $(\nu \vec{y} u)(\Pi R_j \,|\, M_i \,|\, Q') \Downarrow_e [\![\mathbf{b}_i]\!]_w$ $(i = 1,2)$ with $Q' = !w(x).Q$ and $\mathbf{b}_1 = \mathtt{true}$ and $\mathbf{b}_2 = \mathtt{false}$. By Proposition 5.8, we have $\vec{F}$ and F' s.t. $(\lambda \vec{y}.F'x)M_i\vec{F} \Downarrow \mathbf{b}_i$, which contradicts Lemma 5.5, hence done. ■

6. 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 [9] introduced *affine, sequential* types for the π-calculus and established full abstraction for an encoding of PCF. Using causality between names, the present text refines affine, sequential types into *linear* types to ensure strong normalisability and full abstraction for $\lambda_{\rightarrow,\times,+}$. Figure 5 shows the relationship between these results.

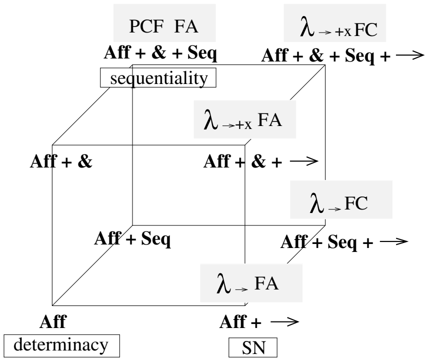

Figure 5. A Family of Affine/Linear Systems

- The addition of branching types is indicated by &, $\rightarrow$ adds causality to action types, and Seq stands for the inclusion of the sequentiality constraints used in [9].
- 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]), while FA stands for *full abstraction*.

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 [9] to $\mathsf{Aff} + \& + \mathsf{Seq}$. Note also that the development in Section 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 [33] to obtain exactly the same result, indicating the flexibility of the proposed calculus to encode functional SN behaviour.

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 truly linear channel x, then an answer is guaranteed to eventually arrive at x, however complex intermediate interaction sequences would be. Below see §4.2 for the notion of closed action types.

Proposition 6.1 (linear honesty) *Let* $\Gamma \vdash x : (\tau)^!$ *be such that* $\mathsf{md}(\tau) = \uparrow$. *Suppose* $\Gamma \vdash P \triangleright A$ *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 6.2 (fundamental lemma for linear actions) *Let* $\Gamma \vdash P \triangleright A \otimes \uparrow x$ *with* A *closed. Then for each* $P \longrightarrow^* P_0$ *there exists* P' *such that* $P_0 \longrightarrow^* P'$ *where* (i) P' *contains no active linear input channels, and* (ii) $P' \stackrel{l}{\longrightarrow}$ *such that* l *is an output at* x.

The proof of Lemma 6.2 uses Theorem 3.1, see Appendix E for detail. Now Proposition 6.1 is immediate by noting, after the mentioned input, the term has the action type mentioned in the Lemma.

We can strengthen Proposition 6.1 by incorporating the possibility that the client itself interacts with the server towards the eventual answer [20]. 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, strengthening preceding related type disciplines, cf. [28, 29, 47].

State and Non-functional Control It is an important subject of study to extend our typing system to allow incorporation of state and non-functional control. 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.

So far we have verified that our proof method is also applicable to SN for first-order stateful processes, albeit under a sequentiality constraint [9]. We foresee no fundamental difficulty in extending the results to concurrent stateful computation, although the lack of the Church-Rosser property makes the reasoning harder.

Complex Causality The present work adds minimum causality to the system in [9] to ensure SN of replicated processes. However, our SN proof seems to be able to cope, without significant change, with more complex causality relations: we could relax the channel type constraints and extend action types to finite graph structures between arbitrary linear nodes as in [47]. 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 this extension, including full proof nets [8]. The status of strong reduction would become subtle in this setting, cf. [14].

Second-order and Other Extensions Can the presented results be augmented to cover more expressive notions of types studied in functional calculi? Adding recursive types [34, 45] easily leads to a system that is not strongly normalising: for example, the encoding, following Figure 5, of $(\lambda x.xx)(\lambda x.xx)$ would be typable. On the other hand, our ongoing work on second-order types suggests that such extensions can coexist harmoniously with SN, as they do in the corresponding functional calculi. We shall report on second-order process types in a subsequent exposition. Other, more refined type structures would also be worth studying in the present context: the use of the π-calculus offers a natural habitat for SN typing systems for stateful, interactive and mobile computation.

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 τ-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 (although the liveness property discussed in Proposition 6.1 closely corresponds to the games-based characterisation of SN). On the other hand, the present work may offer new ways to formulate the notion of SN in game semantics, when acyclicity conditions are explicitly incorporated into game types.

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, 1994. *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] Barendregt, H., *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[7] Barendregt, H., Lambda Calculi with Types, Handbook of Logic in Computer Science, Vol 2, pp.118–310, Clarendon Press, Oxford, 1992.

[8] Bellin, G., and Scott, P.J., On the Pi-calculus and linear logic, *TCS*, Vol. 135, pp. 11-65, 1994.

[9] Berger, M., Honda, K. and Yoshida, N., Sequentiality and the π-Calculus, to appear in *Proc. TLCA01*, LNCS, Springer, 2001. Available at `www.dcs.qmw.ac.uk/ martinb`, 2000.

[10] Boudol, G., Asynchrony and the pi-calculus, INRIA Research Report 1702, 1992.

[11] Boudol, G., The pi-calculus in direct style, *POPL'97*, 228–241, ACM, 1997.

[12] Gallier, J. H., On Girard's "Candidats de Reductibilité", 123–203, Logic and Computer Science, Academic Press Limited, 1990.

[13] Gay, S. and Hole, M., Types and Subtypes for Client-Server Interactions, *ESOP'99*, LNCS 1576, 74–90, Springer, 1999.

[14] Girard, J.-Y., Linear Logic, *TCS*, Vol. 50, 1–102, 1987.

[15] Girard, J.-Y., Lafont Y. and Taylor, P., *Proofs and Types*, vol. 7 of Cambridge Tracts in Theoretical Computer Science, CUP, 1989.

[16] Gunter, C.A., *Semantics of Programming Languages: Structures and Techniques*, MIT Press, 1992.

[17] 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. [38].

[18] Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, 509-523, 1993.

[19] Honda, K., Composing Processes, *POPL'96*, 344-357, ACM, 1996.

[20] Honda, K., Kubo, M. and Vasconcelos, V., Language Primitives and Type Discipline for Structured Communication-Based Programming. *ESOP'98*, LNCS 1381, 122–138. Springer-Verlag, 1998.

[21] Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, 133–147, Springer-Verlag 1991.

[22] Honda, K. Vasconcelos, V., and Yoshida, N. Secure Information Flow as Typed Process Behaviour, *ESOP '99*, LNCS 1782, 180–199, Springer-Verlag, 2000. Full version available as MCS technical report 01/2000, University of Leicester, 2000.

[23] Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *TCS*, 437–486, Vol. 151, North-Holland, 1995.

[24] Huet, G., Confluent reductions: Abstract properties and applications to term rewriting systems, *J., Assoc, Comput. March*, 23(1), 1981, 11-21.

[25] Hyland, M. and Ong, L., "On Full Abstraction for PCF": I, II and III. *Info. & Comp.* 163 (2000).

[26] Igarashi, A. and Kobayashi, N., A generic type system for the pi-calculus, *POPL'01*, ACM, 2001.

[27] Klop, J. W., Term Rewriting Systems, Handbook of Logic in Computer Science, Vol 2, pp.2–117, Clarendon Press, Oxford, 1992.

[28] Kobayashi, N., A partially deadlock-free typed process calculus, *ACM TOPLAS*, Vol. 20, No. 2, 436–482, 1998.

[29] 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.

[30] Kobayashi, N., Pierce, B., and Turner, D., Linear Types and π-calculus, *POPL'96*, 358–371, ACM Press, 1996.

[31] Lafont, Y., Interaction Nets, *POPL'90*, pp. 95–108, ACM press, 1990.

[32] 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.

[33] Milner, R., Functions as Processes. *MSCS*, 2(2), 119–146, CUP, 1992.

[34] Milner, R., Polyadic π-Calculus: a tutorial. *Proceedings of the International Summer School on Logic Algebra of Specification*, Marktoberdorf, 1992.

[35] Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), 1–77, 1992.

[36] Mithcell, J. *Foundations for Programming Languages*, MIT Press, 1996.

[37] Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *LICS'93*, 187–215, IEEE, 1993.

[38] PLAN: A Packet Language for Active Networks, Switch-Ware Project, University of Pennsylvania, available from `http://www.cis.upenn.edu/ switchware/`.

[39] Quaglia, P. and Walker, D., On Synchronous and Asynchronous Mobile Processes, *FoSSaCS 00*, LNCS 1784, 283–296, Springer, 2000.

[40] Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, North-Holland, 1996.

[41] Sangiorgi, D., The name discipline of uniform receptiveness, *ICALP'97*, LNCS 1256, 303–313, Springer, 1997.

[42] Tait, W., Intensional interpretation of functionals of finite type, I. *J. Symb. Log*, 32, 198–212, 1967.

[43] Takahashi, M., Parallel Reductions in Lambda-Calculus, JSC 7(2): 113-123 (1989).

[44] Vasconcelos, V., Typed concurrent objects. *ECOOP'94*, LNCS 821, 100–117. Springer, 1994.

[45] Vasconcelos, V. and Honda, K., Principal Typing Scheme for Polyadic π-Calculus. *CONCUR'93*, LNCS 715, 524–538, Springer-Verlag, 1993.

[46] Winskel, G., *The Formal Semantics of Programming Languages: An Introduction*, MIT Press, 1993.

[47] 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.

Appendix

A. Proofs of Section 2

A.1. Proof of Subject Reduction Theorem

We prove Proposition 2.4 (i). We use the same routine as in [9, 22]. We first show:

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

i. *If $A/\vec{y}$ is defined, then $A/\vec{y}$ is an action type.*
ii. *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 [22]. (i) is trivial. The case $\mathsf{fn}(A_1) \cap \mathsf{fn}(A_2) = \emptyset$ in (ii) is obvious. The other case is proved by the induction on the number of nodes of A_1 and A_2. ■

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

i. (commutativity) *Assume $A_1 \asymp A_2$. Then we have $A_2 \asymp A_1$ and $A_1 \odot A_2 = A_2 \odot A_1$.*
ii. (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) $(A_1 \odot A_2) \odot A_3 = A_1 \odot (A_2 \odot A_3)$.*
iii. *If $px \in |A|$ and $p \in \{!, \downarrow\}$ then x is active in C.*
iv. *If $A \asymp B$ with $A/\vec{x} = A_0$ and $\mathsf{fn}(B) \cap \{\vec{x}\} = \emptyset$, then $A_0 \asymp B$ and $(A \odot B)/\vec{x} = A_0 \odot B$.*
v. *If $A \asymp B$ with $A/\vec{x} = A_0$ and $B/\vec{x} = B_0$, then $A_0 \asymp B$, $A \asymp B_0$, $A_0 \asymp B_0$, and $(A \odot B)/\vec{x} = A_0 \odot B_0$.*

PROOF: (i,ii) are proved as Lemma 3.5 in [22]. (iii) is by definition. (iv,v) by Lemma A.1 and (i-iii). ■

Lemma A.3 i. *$\Gamma \vdash P \triangleright A$ and $P \equiv Q$ then $\Gamma \vdash Q \triangleright A$.*

ii. *$\Gamma \vdash x(\vec{y}{:}\vec{\tau}).P \,|\, \overline{x}(\vec{y}{:}\vec{\tau}')Q \triangleright A$ implies $\Gamma \vdash (\nu \vec{y}{:}\vec{\alpha})(P \,|\, Q) \triangleright A$ with $\alpha_i = \langle \tau_i,\, \overline{\tau_i} \rangle$ and $\overline{\tau_i} = \tau'_i$.*
iii. *$\Gamma \vdash !x(\vec{y}{:}\vec{\tau}).P \,|\, \overline{x}(\vec{y}{:}\vec{\tau}')Q \triangleright A$ implies $\Gamma \vdash !x(\vec{y}{:}\vec{\tau}).P \,|\, (\nu \vec{y}{:}\vec{\alpha})(P \,|\, Q) \triangleright A$ with $\alpha_i = \langle \tau_i,\, \overline{\tau_i} \rangle$ and $\overline{\tau_i} = \tau'_i$.*

PROOF: The proof is essentially the same as in [9, 22]. Assume $\Gamma \vdash P \triangleright A$. As in [9, 22], there exists the minimum action type A_0 such that $A_0 \subseteq A_1$ and $\Gamma \vdash P \triangleright A_0$. Hence in the following we only consider the minimum action types.

For (i), 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 $\Gamma \vdash (\overline{x}(\vec{y})P)|Q \triangleright A$ such that $x \notin \mathsf{fn}(Q)$. By typing rules we can assume $\Gamma \vdash \overline{x}(\vec{y})P \triangleright A_1$ and $\Gamma \vdash Q \triangleright A_2$ 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 $\Gamma \vdash \overline{x}(\vec{y})P \triangleright A_1$ we deduce $\Gamma \cdot \vec{y}{:}\vec{\tau} \vdash P \triangleright C$ with $C/\vec{y} \odot ?x = A_1$. By $(C/\vec{y} \odot ?x) \asymp A_2$, using (iv) in Lemma A.2 and associativity, we have $(C \odot ?x)/\vec{y} \asymp A_2$ and $(C \odot ?x)/\vec{y} \odot A_2 = A$, so that $\Gamma \vdash \overline{x}(\vec{y})(P|Q) \triangleright A$. The inverse and other cases are similarly reasoned (see [9, 22]).

(ii) is just the same as in [22] by restricting the maximum length of ordering between nodes to 1.

For (iii), we prove the monadic case. Polyadic case is just the same. Suppose

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

Then, we have the derivations such that

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

with $A_1 \stackrel{\text{def}}{=} (!x \to ?\vec{z})$ for some $\vec{z}$, $A_1 \odot A_2 = A$ and $A_2 = A_3 \odot ?x$. Then the above input process is derived by $(\mathsf{In}^!)$ from

$$\Gamma \cdot y{:}\tau \vdash P \triangleright A_3 \quad \text{with } A_3/y = ?\vec{z} \text{ and } \Gamma \vdash x{:}(\tau)^!$$

Similarly by $(\mathsf{Out}^?)$, the output is derived by

$$\Gamma \cdot y{:}\tau' \vdash Q \triangleright A_4 \quad \text{with } A_4/y = A_2 \text{ and } \Gamma \vdash x{:}(\tau')^?$$

First, by $\Gamma \vdash x{:}(\tau)^!$ and $\Gamma \vdash x{:}(\tau')^?$, we have $\overline{\tau} = \tau'$. Suppose $\mathsf{md}(\tau) = ?$. Then we can write $A_3 = ?\vec{z}_2 \otimes ?y$ with $\vec{z}_2 y = \vec{z}$, $A_4 = !y \to ?\vec{w} \otimes A_5$ and $A_2 = ?\vec{w} \otimes A_5$ for some $\vec{w}$ and A_5. Hence we can check $A_3 \asymp A_4$ by $A_1 \asymp A_2$. Thus by (Par), we have

$$\Gamma \cdot y{:}\alpha \vdash P \,|\, Q \triangleright A_3 \odot A_4 \quad \text{with } \alpha = \langle \tau,\, \overline{\tau} \rangle$$

By (v) in Lemma A.2, $(A_3 \odot A_4)/y = A_1/x \odot A_2/x$. Hence by $!y \in A_3$, we first apply (Res) to have:

$$\Gamma \vdash (\nu y{:}\alpha)(P \,|\, Q) \triangleright A_1/x \odot A_2/x$$

Next we apply $(\mathsf{Weak}\text{-}?)$ to A_2/x in order to obtain A_2. Then we have:

$$\Gamma \vdash (\nu y{:}\alpha)(P \,|\, Q) \triangleright A_1/x \odot A_2$$

By $A_1 \odot A_1/x = A_1$, together with associativity, finally we have

$$\Gamma \vdash !x(y{:}\tau).P \,|\, (\nu y{:}\alpha)(P \,|\, Q) \triangleright A$$

The case $\mathsf{md}(\tau) = \uparrow$ is just the same. ■

A.2. Proof of Strong Confluence

We prove Proposition 2.4 (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 $R \longrightarrow (\nu\vec{c})(!a(\vec{x}).P \,|\, (\nu\vec{x})(P \,|\, Q_1) \,|\, \overline{a}(\vec{x})Q_2 \,|\, Q') \stackrel{\text{def}}{=} R_1$ and $R \longrightarrow (\nu\vec{c})(!a(\vec{x}).P | \overline{a}(\vec{x})Q_1 \,|\, (\nu\vec{x})(P \,|\, Q_2) \,|\, Q') \stackrel{\text{def}}{=} R_2$. 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 $R_2 \longrightarrow Q'$.

B. Proofs for Section 3

B.1. Proof of Proposition 3.3

(i) is the just same as the proof of $\longrightarrow$. For (ii), 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 i. (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, $P \mapsto_{\mathsf{r}} S \mapsto_{\mathsf{g}}^{+} R$.*

ii. (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 $Q_i \mapsto_{\mathsf{g}} R$.*

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

iv. (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 $Q_i \mapsto_{\mathsf{l}} R$.*

v. (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. [27]). 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 [6] based on the labelled reduction [32]. 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. [32] and Section 11.2 in [6]). Let P_1 be the result of replacing the redex $\overline{a}(\vec{y})Q_1$ in P by its reduct $(\nu\vec{y})(R_1 \,|\, Q_1)$. 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 [32]. Consequently, if we underline a in $\overline{a}(\vec{y})Q_1$, we only need to reduce all occurrences of the underlined a in P_2 to obtain P_3. The rest is the just same as in [6]. ∎

B.2. Well-definedness of $[\![\Gamma,\ A]\!]$.

The rules in Figure 3 (defining $[\![\Gamma,\ A]\!]$) are well-defined using the ordering $\lneq$ over the pairs of form Γ, A (with $\Gamma \vdash A$) generated as follows. Below we use the *height* of a pair Γ, A given by $\sharp(\Gamma, A) \stackrel{\text{def}}{=} \sum_{x \in \mathsf{fn}(A)} \Gamma(x)$, where $\sharp((\vec{\tau})^{p}) \stackrel{\text{def}}{=} \sum_i \sharp(\tau_i)$ and $\sharp(\langle\tau, \vec{\tau}\rangle) \stackrel{\text{def}}{=} \sharp(\tau)$.

i. If $\sharp(\Gamma, A) \lneq \sharp(\Delta, B)$ then $\langle\Gamma, A\rangle \lneq \langle\Delta, B\rangle$.

ii. If $\overline{p}x \in |A|$ with $p \in \{\downarrow, !\}$ then $\sharp(\Gamma, A \odot px) \lneq \sharp(\Gamma, A)$.

Based on this ordering (which is applied to all rules including those for prime semantic types) we can verify the rules give the definition by induction based on well-ordering (cf.[46]).

B.3 Typing Rules for Prime Prefixes

In the proofs we use the variant of the typing rules for output prefixes which are restricted to primes.

$$\frac{(\mathsf{Out}^{\uparrow})\ \ (C/\vec{y} = \uparrow A) \quad \Gamma \vdash x : (\vec{\tau})^{\uparrow} \quad \mathsf{active}(C) = \{\vec{y}\} \quad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \triangleright C^{-x}}{\Gamma \vdash \overline{x}(\vec{y} : \vec{\tau})P \triangleright A \otimes \uparrow x}
\qquad
\frac{(\mathsf{Out}^{?})\ \ (C/\vec{y} = \uparrow ?A) \quad \Gamma \vdash x : (\vec{\tau})^{?} \quad \mathsf{active}(C) = \{\vec{y}\} \quad \Gamma \cdot \vec{y} : \vec{\tau} \vdash P \triangleright C}{\Gamma \vdash \overline{x}(\vec{y} : \vec{\tau})P \triangleright A \odot ?x}$$

With the same proof as in Appendix D of [9], we can easily see:

Proposition B.3 *If $\Gamma \vdash P \triangleright A$ in the system in Figure 2, then $\Gamma \vdash P_0 \triangleright A$ for $P_0 \equiv P$ in the system with the same rules except output rules are replaced by the above rules. In particular, we can assume (up to $\equiv$) active names under an output prefix to be bound by that prefix.*

Thus we can assume, without loss of generality, that all processes are prime when discussing properties which are invariant under $\equiv$ (such as strong normalisability).

B.4 Omitted Proofs for Lemma 3.6

We prove Lemma 3.6 by induction on structure of terms following the construction of NF_e.

(Linear Input). Assume $P \in [\![\Gamma \cdot \vec{y} : \vec{\tau},\ C]\!]$ with $C/\vec{y} = \uparrow A \otimes ?B$. We show $x(\vec{y} : \vec{\tau}).P \in [\![\Gamma,\ (\downarrow x \to \uparrow A) \otimes ?B]\!]$. First we note $\mathsf{cl}((\downarrow x \to A) \otimes B) = \overline{A} \otimes \overline{B}$. Let $Q \in [\![\Gamma,\ \overline{A} \otimes \overline{B}]\!]$. W.l.o.g. we assume $Q \equiv Q_1 | Q_2$ such that $Q_1 \equiv \Pi_i Q_{1i\langle\downarrow a_i\rangle}$ and $Q_2 \equiv \Pi_j Q_{2j\langle !b_j\rangle}$. We also note that $\mathsf{md}(\tau_i) = ?$ for each i by the typing rule. By (IH) $P|(Q|R) \Downarrow_e Q_2|R$ for any $R \in [\![\Gamma \cdot \vec{y} : \vec{\vec{\tau}},\ !y_i]\!]$, hence 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 $P' \in [\![\Gamma \cdot \vec{y} \cdot \vec{\tau},\ \otimes ?y_i]\!]$, which implies, by construction, $x(\vec{y}).P' \in \langle\langle\Gamma,\ \downarrow x\rangle\rangle$. So take any $R \in [\![\Gamma\cdot\vec{y} : \vec{\overline{\tau}},\ \otimes !y_i]\!]$. W.l.o.g. we set $R \in \mathsf{NF}_e$. We already know $P|Q|R \mapsto^* P'|Q_2|R$, while by (IH) we have $P|Q|R \Downarrow_e Q_2|R$. Hence by CR of $\mapsto$ we know $P'|Q_2|R \Downarrow_e Q_2|R$. This shows $P' \in [\![\Gamma\cdot\vec{y}\cdot\vec{\tau},\ \otimes ?y_i]\!]$.

(Replicated Input). Similar to the previous case.

(Linear Output). Assume $P \in [\![\Gamma\cdot\vec{y} : \vec{\tau},\ C]\!]$ with $C/\vec{y} = ?B$. We show that $\overline{x}(\vec{y} : \vec{\tau})P \in [\![\Gamma,\ \uparrow x \otimes B]\!]$. Note $\mathsf{cl}(\uparrow x \otimes B) = \downarrow x \otimes \overline{B}$. Let $Q \in [\![\Gamma,\ \downarrow x \otimes \overline{B}]\!]$. W.l.o.g. $Q \equiv x(\vec{y}).Q'_0 \,|\, \Pi_j Q_{j\langle !b_j\rangle}$. Then we have:

$$\overline{x}(\vec{y})P \,|\, x(\vec{y}).Q'_0 \,|\, \Pi_j Q_{j\langle !b_j\rangle} \longrightarrow (\nu\,\vec{y})(P|Q'_0) \,|\, \Pi_j Q_j.$$

By $\mathsf{cl}(C) = \overline{B}$ and $\Pi_j Q_j \in [\![\Gamma,\ \overline{B}]\!]$ as well as $P \in [\![\Gamma\cdot\vec{y} : \vec{\tau},\ C]\!]$, we have $(P|\Pi_j Q_j) \Downarrow_e (P'|\Pi_j Q_j)$ where $P' \equiv \Pi_k P'_{k\langle !y_k\rangle}$. By assumption $Q'_0 \in [\![\Gamma\cdot\vec{y} : \vec{\overline{\tau}},\ \otimes ?y_k]\!]$, hence $Q'_0|P' \Downarrow_e P'$. Thus we have:

$$\begin{array}{rcl}(\nu\,\vec{y})(P|Q'_0) \,|\, \Pi_j Q_j & \mapsto^* & (\nu\,\vec{y})(P'|Q'_0) \,|\, \Pi_j Q_j \\ & \mapsto^* & (\nu\,\vec{y})P' \,|\, \Pi_j Q_j \\ & \mapsto^*_{\mathsf{g}} & \Pi_j Q_j.\end{array}$$

(Composition of Primes). Given $\Pi_{i\in I} P_i$, assume by induction $P_i \in [\![\Gamma,\ A_i]\!]$ $(i \in I)$ 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 $?x \in |A_i| \cap |A_j|$. Let $C = A_1 \odot .. \odot A_n$ and $Q \in [\![\Gamma,\ \mathsf{cl}(C)]\!]$. 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 (IH) (the first part), we have $P_i|Q \Downarrow_e P'_i \,|\, \Pi_{h\neq i}(\Pi_{k_h} Q'_{k_h})$ with $P'_i \in [\![\Gamma,\ A_i\cdot\mathsf{cl}(A_i)]\!]$ for each i. Since only replicated Q''_j is shared among P_i, we have $\Pi P_i \,|\, Q \;\Downarrow_e\; \Pi P'_i \,|\, \Pi_j Q''_{j\langle !b_j\rangle}$, as required. We have now exhausted all cases. ■

C Proofs for Section 4

In this section we shall provide some more details towards the proof of Theorem 4.4.

C.1 Proofs of Preliminary Lemmas

Lemma 4.2 follows from the strong normalisability of the calculus (Theorem 3.1) by induction on the derivation of $\Gamma \vdash P \triangleright A$.

Lemma C.1 *If $\Gamma \vdash P \triangleright A$ is a $\triangleright$-normal form and $P \xrightarrow{l} Q$ is an* untyped *transition, the $l \neq \tau$.*

PROOF: By induction on the derivation of $P \xrightarrow{l} Q$ we show that $l = \tau$ would imply that P is not in NF_e, hence not a $\triangleright$-normal form. ■

Lemma C.2 *If $\Gamma \vdash P \triangleright A$ and P is a $\triangleright$-normal form then $P \equiv' P_\downarrow|P_\uparrow|P_!|P_?$ where*

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

Here $I_\downarrow, I_\uparrow, I_!, I_?$ partition the finite set I such that (1) for all $i, j \in I \setminus I_?$: $i \neq j$ implies $x_i \neq x_j$, (2) for all $i \in I_! \cup I_?$ and all $j \in I_\downarrow \cup I_\uparrow$: $x_i \neq x_j$ and (3) P_i is in $\triangleright$-normal form for all $i \in I$. Furthermore, $P_\downarrow$, $P_\uparrow$, $P_!$ and $P_?$ are unique up to $\equiv'$.

PROOF: By induction on the derivation of $\Gamma \vdash P \triangleright A$. ■

Definition C.3 *With P as in Lemma C.2, we call $P \equiv' P_\downarrow|P_\uparrow|P_!|P_?$ a* $\triangleright$-normal form decomposition *of P with* $\downarrow$-component $P_\downarrow$, $\uparrow$-component $P_\uparrow$, $!$-component $P_!$ *and* $?$-component $P_?$.

Lemma C.4 *Let $\Gamma \vdash P^i \triangleright A$ have a $\triangleright$-normal form decomposition $P^i = P^i_\downarrow|P^i_\uparrow|P^i_!|P^i_?$ for $i = 1, 2$.*

i. *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 σ of $\{1,\dots,n\}$ 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.*
ii. *Assume that $P^1_\uparrow = \Pi^m_{j=1} \overline{y_i}(\vec{z}_i).P^1_j$ and $P^2_\uparrow = \Pi^n_{j=1} \overline{a_i}(\vec{b}_i)P^2_j$. Then $P^1_\uparrow \approx P^2_\uparrow$ iff $m = n$ and there is a permutation σ of $\{1,\dots,n\}$ such that $\overline{y_i}(\vec{z}_i)P^1_j \approx \overline{a_{\sigma(i)}}(\vec{b_{\sigma(i)}})P^2_{\sigma(i)}$ for all i.*
iii. *Assume that $P^1_! = \Pi^m_{j=1} !y_i(\vec{z}_i).P^1_j$ and $P^2_! = \Pi^n_{j=1} !a_i(\vec{b}_i).P^2_j$. Then $P^1_! \approx P^2_!$ iff $m = n$ and there is a permutation σ of $\{1,\dots,n\}$ 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.*
iv. *Assume that $P^1_? = \Pi^m_{j=1} \overline{y_i}(\vec{z}_i).P^1_j$ and $P^2_? = \Pi^n_{j=1} \overline{a_i}(\vec{b}_i)P^2_j$. Then $P^1_? \approx P^2_?$ iff $m = n$ and there is a permutation σ of $\{1,\dots,n\}$ such that $\overline{y_i}(\vec{z}_i)P^1_j \approx \overline{a_{\sigma(i)}}(\vec{b_{\sigma(i)}})P^2_{\sigma(i)}$ for all i.*
v. *If $\Gamma \vdash P^i \triangleright A$ $(i = 1, 2)$ has $\triangleright$-normal form decomposition $P^i_\downarrow|P^i_\uparrow|P^i_!|P^i_?$ for $i = 1, 2$ then $P^1 \approx P^2$ 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: (i, ii, iii) are immediate by considering traces.

For (iv) we proceed by contradiction. Assume for the non-trivial part of the proof that 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 B.3, 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 C.1, no process in $\triangleright$-normal form can have a τ-transition. Hence $P^1_?$ and $P^2_?$ cannot have the same set of traces.

Finally, (v) follows from (i, ii, iii, iv). ■

Lemma C.5

i. *If* $\mathbf{R}$ *is a bisimulation up to* $\equiv$ *then it is a bisimulation.*

ii. *If* $\mathbf{R}$ *is a bisimulation then so is the symmetric closure of* $\mathbf{R}$.

iii. $\equiv'$ *is a bisimulation.*

iv. *If the context* $C[\,]$ *in the definition of extended reduction (Definition 3.2) is a reduction context, then both,* $\mapsto_{\mathsf{l}}$ *and* $\mapsto_{\mathsf{r}}$*, can be replaced by* τ*-transitions.*

PROOF: By induction on the structure of $C[]$ we show (iv). The other statements are standard. ■

C.1.1 Proof of Lemma 4.3

Proposition C.6 *Let us define the* size *of a process* P, $\mathsf{size}(P)$*, as the number of constructors in* P*. Then* $\mathsf{size}(P)$ *is invariant under* $\equiv'$.

PROOF: By induction on $\mathsf{size}(P)+\mathsf{size}(Q)$ we show that $P \approx Q$ implies $P \equiv' Q$, the reverse being Lemma C.5 (iii). The base case, $\mathsf{size}(P)+\mathsf{size}(Q) = 2$ is immediate. The inductive step uses the (IH) together with Lemma C.4. ■

C.2 Proof of Theorem 4.4

PROOF: For the first equation in (i) we establish $\mathbf{R}\cup\mathbf{id}$ to be a typed bisimulation up to $\equiv$ and apply Lemma C.5 (i), where $\mathbf{R}$ is inductively given by the following rules, assuming, for each rule instance $P_1\mathbf{R}P_2$, the existence of a base Γ and action type A such that $\Gamma \vdash P_i$ for $i = 1,2$.

$$\begin{array}{rcl} C[\overline{x}(\vec{y})P]|x(\vec{y}).Q & \mathbf{R} & C[(\nu\vec{y})(P|Q)] \\ C[\overline{x}(\vec{y})P]|!x(\vec{y}).Q & \mathbf{R} & C[(\nu\vec{y})(P|Q)]\,|\,!x(\vec{y}).Q \\ (\nu x)!x(\vec{y}).Q & \mathbf{R} & \mathbf{0} \end{array}$$

Here $C[]$ is an arbitrary context not binding x. It is clear that $\mathbf{R}\cup\mathbf{id}$ is a typed relation. To see that it is also a bisimulation, let $R = C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{l} R'$, $\Gamma \vdash R \triangleright A$ and $\Gamma \vdash R \xrightarrow{l} R'$. Then we can have the following *untyped* transitions causing the *typed* transition above (the interacting parts are highlighted):

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. $C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{x(\vec{y})} C[\overline{x}(\vec{y})P]|Q$ and

6. $C[\overline{x}(\vec{y})P]|x(\vec{y}).Q \xrightarrow{\overline{x}(\vec{y})} C[P]|x(\vec{y}).Q$.

We shall now establish that only the processes in (1), (2) and (3) are typable. To this end we show by induction on the derivation of $\Gamma \vdash R \triangleright A$ that $\bot x \in A$. 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.

It is clear that 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) up to $\equiv$ 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). If $\Gamma \vdash C[\overline{x}(\vec{y})P]|!x(\vec{y}).Q \xrightarrow{l} R'$ we proceed similarly.

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

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. $C[(\nu\vec{y})(P|Q)] \xrightarrow{l} C'[(\nu\vec{y})(P|Q')]$

Clearly, (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 proceed by induction on the transition to show that $C[\,]$ must be a reduction context. Then we can apply Lemma C.5 (iv) to 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.

By induction on the length of extended reduction sequences we can now prove $\mapsto^* \subseteq \approx$ from $\mapsto \subseteq \approx$. The strictness of the inclusions is immediate by noting that $\approx$ is symmetric, unlike $\mapsto$ and $\mapsto^*$. ■

D Proofs for Section 5

D.1 Reduction in $\lambda_{\to,\times,+}$

$\rightsquigarrow$ is the least compatible relation including:

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

Other possible notions of reduction include commuting conversions and η-rules [14].

D.2 Typing and Reduction Rules for Branching

To be able to type the branching constructs, we add:

$$\frac{(\mathsf{Bra}^!)\ (C_i/\vec{y}_i = ?B) \quad \Gamma \vdash x : [\&_{i\in I}\vec{\tau}_i]^! \quad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash P_i \triangleright C_i^{-x}}{\Gamma \vdash x[\&_{i\in I}(\vec{y}_i : \vec{\tau}_i).P_i] \triangleright !x \to B}
\qquad
\frac{(\mathsf{Sel}^?)\ (C_i/\vec{y}_i = A \asymp ?x) \quad \Gamma \vdash x : [\oplus_{i\in I}\vec{\tau}_i]^? \quad \Gamma\cdot\vec{y}_i : \vec{\tau}_i \vdash P \triangleright C}{\Gamma \vdash \overline{x}\mathtt{in}(\vec{y}_i : \vec{\tau}_i)P \triangleright A \odot ?x}$$

$(\mathsf{Bra}^{\downarrow})$ and $(\mathsf{Sel}^{\uparrow})$ are defined similarly. The reductions for branching are given by:

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

$$(\textsc{Bra}_!) \quad !x[\&_i(\vec{y}_i).P_i] \,|\, \overline{x}\mathtt{in}_i(\vec{y}_i)Q \longrightarrow !x[\&_i(\vec{y}_i).P_i] | (\nu\,\vec{y}_i)(P_i|Q)$$

The rules for extended reductions are similarly augmented:

$$\begin{array}{rcl}
C[\overline{x}\mathtt{in}_j(\vec{y}_j)P]|x[\&_i(\vec{y}_i).Q_i] & \mapsto_{\mathsf{l}} & C[(\nu\,\vec{y}_j)(P|Q_j)] \\
C[\overline{x}\mathtt{in}_j(\vec{y}_j)P]|!x[\&_i(\vec{y}_i).Q_i] & \mapsto_{\mathsf{r}} & C[(\nu\,\vec{y}_j)(P|Q_j)]|!x[\&_i(\vec{y}_i).Q_i] \\
(\nu x)!x[(\vec{y})_i.Q_i] & \mapsto_{\mathsf{g}} & \mathbf{0}
\end{array}$$

Labelled transitions use additional labels, $x\mathtt{in}_i(\vec{y})$ and $\overline{x}\mathtt{in}_i(\vec{y})$.

$$(\textsc{Bra}) \quad x[\&_{i\in I}(\vec{y}_i).P_i] \stackrel{x\mathtt{in}_j(\vec{y}_j)}{\longrightarrow} P_j$$

$$(\textsc{Bra}_!) \quad !x[\&_{i\in I}(\vec{y}_i).P_i] \stackrel{x\mathtt{in}_j(\vec{y}_j)}{\longrightarrow} !x[\&_{i\in I}(\vec{y}_i).P_i]|P_j$$

$$(\textsc{Sel}) \quad \overline{x}\mathtt{in}_i(\vec{z})\cdot P \stackrel{\overline{x}\mathtt{in}_i(\vec{z})}{\longrightarrow} P$$

D.3 Proof of Lemma 5.1

The following result is proved precisely as in [9].

Proposition D.1 *$\Gamma \vdash P_1 \cong_{\mathsf{sn}} P_2$ iff $\Gamma \cup \Delta \vdash P_1|R \Downarrow_x^i \Leftrightarrow P_2|R \Downarrow_x^i$ for each composable R.*

Based on this we now prove a result stronger than Lemma 5.1. Below given A such that $|A| = \{p_i x_i\}_{p_i \neq \perp} \cup \{\perp y_j\}$ we define $\overline{A} \stackrel{\text{def}}{=} \otimes_i \overline{p}_i x_i$.

Proposition D.2 *Let Γ be paired. Then $\Gamma \vdash P_1 \cong_{\mathsf{sn}} P_2 \triangleright A$ iff $\Gamma \vdash P_1|R \Downarrow_x^i \Leftrightarrow P_2|R \Downarrow_x^i$ for each R such that $\Gamma, x : [\oplus_{1,2}]^{\uparrow} \vdash R \triangleright \overline{A} \otimes \uparrow x$.*

PROOF: The "then" direction is trivial, For the "if" direction, assume the r.h.s. holds. Let R be composable to P and $\Gamma' \vdash P_1|R \Downarrow_x^i$. By the definition of convergence, we can set $\Gamma' \vdash R \triangleright \overline{A} \otimes !B \otimes \perp C \otimes \uparrow x$ (note that even when A contains a linear output channel, an observable at that channel can always be mediated by a copy-cat, so this loses no precision). By Theorem 3.9 and CR, assume w.l.o.g. $R \in \mathsf{NF}_e$. Then we know $R \equiv R_0|R'$ where $\Gamma' \vdash R_0 \triangleright \overline{A} \otimes \uparrow x$ and $\Gamma' \vdash R' \triangleright !B$ (note all $\perp$-channels disappear by being in NF_e). But $\mathsf{fn}(B) \cap (\mathsf{fn}(A) \cup \{x\} = \emptyset$ by definition, so that $P_1|R_0|R' \Downarrow_x^i$ iff $P_1|R_0 \Downarrow_x^i$. By assumption this implies $P_2|R_0 \Downarrow_x^i$, hence $P_2|R \Downarrow_x^i$ as required. ■

Now Lemma 5.1 is an easy corollary of Proposition D.2.

D.4 Proof of Lemma 5.2

Below we assume all outputs are primes without loss of precision, and, for brevity, write L for $\overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).(\overline{x}\mathtt{in}_l(\vec{y})P|R_i)])$ (the l.h.s. of the equation) and R for $(\overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).R_i]) \mid \overline{x}\mathtt{in}_l(\vec{y})P$ (the r.h.s. of the equation). Since all free occurrences of channels under ?-prefix should be output by the typing rules, we safely assume:

$$\Gamma \vdash L \triangleright ?w \otimes \uparrow x \otimes ?A \otimes \uparrow B$$

By Proposition D.2 we compose this with S s.t.

$$\Gamma \vdash S \triangleright !w \otimes !A \otimes \downarrow B \otimes \downarrow x \otimes \uparrow z.$$

Now by Theorem 3.1 we know $L|S \Downarrow_e O$ for some $O \in \mathsf{NF}$. By Lemma 6.2, we know S' does not contain any active linear input. This in particular means that an input at x is gone, so that there is some T' such that

$$L|S \mapsto^* S' \mapsto S'' \mapsto^* O$$

where S' does not contain an active output at x while S'' does so. Since no reduction involves the channel x until $S' \mapsto S''$, each reduction until S' can be exactly followed by $R|S$ (this is easily established by forming a closure), we now know

$$R|S \mapsto^* S'$$

by which we know $R|S \Downarrow_e O$, giving the same observable. ■

For reference we list two special cases of the equations. Assume below the type of x is unary, $\overline{x}(\vec{y})P$ is prime and w has mode ? below.

$$\begin{array}{lll}
\Gamma \vdash & \overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).(\overline{x}(\vec{y})P|R_i)]) & \cong_{\mathsf{sn}} \\
& \overline{w}(\vec{a}b\vec{c})(Q \mid b[\&_i(\vec{z}).R_i]) \mid \overline{x}(\vec{y})P & \\
\Gamma \vdash & \overline{w}(\vec{a}b\vec{c})(Q \mid b(\vec{z}).(\overline{x}(\vec{y})P|R)) & \cong_{\mathsf{sn}} \\
& \overline{w}(\vec{a}b\vec{c})(Q \mid b(\vec{z}).R]) \mid \overline{x}(\vec{y})P &
\end{array}$$

Note non-unary linear output prefixes may occur in distinct branches as different injections. This never happens with unary linear outputs, so they can always be pulled out as shown above.

D.5 η-expansion

In the definability result we use the following observation.

Proposition D.3 (η-expansion) *Assume that $!u(\vec{x}z\vec{y}).P$ is typable with z typed as $(\tau)^{\uparrow}$. Then we have $!u(\vec{x}z\vec{y}).P \cong_{\mathsf{sn}} !u(\vec{x}z\vec{y}).\overline{z}(m)P'$ for some m and P'.*

The transformation from the left to the right is called η-*expansion*. For the proof we first note the following variant of the permutation lemma.

Lemma D.4 *Assume $\overline{w}(c)c[\&_i(\vec{y}_i).\overline{z}(k)!k(\vec{v})P_i]$ is typable with z typed as $\uparrow$ and w and other free names as ?. Then we have:*

$$\overline{w}(c)c[\&_i(\vec{y}_i).\overline{z}(k)!k(\vec{v}).P_i] \cong_{\mathsf{sn}} \overline{z}(k)!k(\vec{v}).\overline{w}(c)c[\&_i(\vec{y}_i).P_i]$$

PROOF: By Proposition D.2 we consider contexts which complements these two processes. Such a context contains processes of the forms $z(k).R$ and $!w(c).S$ and other input processes. By repeated extended reductions with the remaining context, we can eliminate all free outputs in $!u(c).S$ so that we can safely assume the only free name in S is c, thus we can set $S \stackrel{\text{def}}{=} \overline{c}\mathtt{in}_l(\vec{y}_l)S'$. Writing Q_1 and Q_2 for the l.h.s. and r.h.s. of the required equation respectively, we have:

$$Q_i \mid z(k).R \mid !w(c).S \longrightarrow^* (\nu k)(!k(\vec{v})P_l|R|S') \quad (i=1,2).$$

Thus we know any resulting observables (composed with other processes as necessary) can only be identical. ■

By repeatedly applying Lemma D.4, we obtain Proposition D.3.

D.6 Sequential Processes

The sequential version of the linear type system precisely follows [9] except for the usage of linear action types. The sequents are of the form $\Gamma \vdash_\phi P \triangleright A$ where $\phi \in \{\mathtt{I},\mathtt{O}\}$ is an *IO-mode*. ϕ obeys the partial algebra $\mathtt{I} \odot \mathtt{I} = \mathtt{I}$ and $\mathtt{I} \odot \mathtt{O} = \mathtt{O} \odot \mathtt{I} = \mathtt{O}$, which, when used in (Par), ensures single threading. An output (resp. input) can only prefix a body in input (resp. output) mode, resulting in output (resp. input) mode. Weakening and hiding do not change IO-modes. An insignificant change from [9] is that $\mathbf{0}$ can be given any IO-mode. This is necessary for SN processes since we allow threads to disappear. We can safely incorporate this relaxation into [9] without changing the set of typed terms modulo the typed bisimilarity given in [9] (the more stringent original rule is used in [9] for the semantic properties of typed processes to be better reflected in their syntactic structures).

D.7 Proof of Lemma 5.7

In the proof of Lemma 5.7 we use the existence of a semantically equivalent sequential process for each linear process. We give here the inductive transformation which achieves this sequentialisation. We call a process a $\lambda_{\to,\times,+}$-*processes* if it is in the form $E^\circ, u : T^\circ \vdash P \triangleright !u \otimes ?A$, with $P \in \mathsf{NF}_e$. As Q in Lemma 5.7 is indeed a subterm of such a process, this is sufficient for the proof of Lemma 5.7. Below an output process is *active* if it is not under any input prefix.

Fact D.5 *Let P be a subterm of a $\lambda_{\to,\times,+}$-process. Then P does not contain two distinct active linear outputs.*

PROOF: By contradiction. Assume R is a $\lambda_{\to,\times,+}$-processes and P is a subterm of a $\lambda_{\to,\times,+}$-process which contains two active linear outputs, say at x_1 and x_2. Since R has no free linear output, x_1 and x_2 are abstracted by some replicated input prefix in R. Since each such prefix abstracts a unique name, one of them is abstracted first. But this is impossible by the typing rule. ■

The rules of transformation into sequential processes follow (cf. Lemma 5.7), which are given for branching prefixes which subsume those for unary ones.

$$\begin{array}{rcl}
\mathbf{0}^\circ & \stackrel{\text{def}}{=} & \mathbf{0} \\
(P|Q)^\circ & \stackrel{\text{def}}{=} & (P^\circ) \mid_{\mathsf{seq}} (Q^\circ) \\
(\overline{a}\mathtt{in}_i(\vec{b})P)^\circ & \stackrel{\text{def}}{=} & \overline{a}\mathtt{in}_i(\vec{b})(P^\circ) \\
(a[\&_i(\vec{b}_i).P_i])^\circ & \stackrel{\text{def}}{=} & a[\&_i(\vec{b}_i).(P_i^\circ)] \\
(!a[\&_i(\vec{b}_i).P_i])^\circ & \stackrel{\text{def}}{=} & !a[\&_i(\vec{b}_i).(P_i^\circ)]
\end{array}$$

where, for sequential P and Q such that $\Gamma \vdash P|Q$, we define $P \mid_{\mathsf{seq}} Q$ by induction on the size of terms as follows. Below we say a sequential P is in *I-mode* (resp. in *O-mode* if it contains no (resp. a unique) active output.

i. If one (or both) of P and Q is (are) in I-mode, then $P \mid_{\mathsf{seq}} Q \stackrel{\text{def}}{=} P|Q$.
ii. If both are in O-mode, then, by Fact D.5, at least one is a ?-channel. Let w in P be such a channel (with the corresponding prime term P_0 s.t. $P \equiv P_0|P'$) and x be another active output channel in Q (with the corresponding prime term Q_0 s.t. $Q \equiv Q_0|Q'$). Then we can "tuck" Q into P using Lemma 5.2, where we use $\mid_{\mathsf{seq}}$ for composition of R_i in Lemma 5.2 and Q_0, which are both sequential and which is well-defined because term sizes strictly decreases. Call the result P_0' and then define $P \mid_{\mathsf{seq}} Q \stackrel{\text{def}}{=} P_0'|P'|Q'$.

It is now easy to check that the map inductively generates a sequential process $\cong_{\mathsf{sn}}$-equivalent to the original process.

D.8 Canonical Normal Forms

Canonical normal forms 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). Their operational semantics and typing rules are standard. As an example of evaluation, in (let-prod), N is first evaluated until it has shape $\langle N, N'\rangle$ and, when it does so, we substitutes them for x and y in M, and continue evaluation. Further, we can translate CNFs to the original syntax. A simplest translation is by transforming "let" into the substitution:

$$\begin{array}{rcl}(\texttt{let}\ ()=z\ \texttt{in}\ F)^\circ &=& F^\circ,\\ (\texttt{let}\ x=zF\ \texttt{in}\ F')^\circ &=& (F')^\circ\{zF^\circ/x\},\\ (\texttt{let}\ \langle x,y\rangle=z\ \texttt{in}\ F)^\circ &=& F^\circ\{\pi_1(z),\pi_2(z)/x,y\}\end{array}$$

Other constructors are identically mapped. This translation has a property that the image covers all and only β-normal forms of $\lambda_{\rightarrow,\times,+}$. Using the same definition for observational congruence, the added constructs in their own operational semantics are equivalent to their translation into the original syntax, so that they add nothing to $\lambda_{\rightarrow,\times,+}$ semantically.

By combining the above translation and $[\![\ ]\!]_u$, we already own the encoding of CNFs into the π-calculus. However, for the proof of Proposition 5.8, it is more convenient to use the following direct encoding of CNFs into processes. This translation maps CNFs directly to extended normal forms. In D.9 we shall show how CNFs exhaust all semantic points in our process representation via this mapping. Below we omit type annotations. For the lets, we assume $[\![F]\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P$. In the case, we assume $[\![F_i]\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P_i$.

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

We note:

Proposition D.6 $[\![F:\alpha]\!]_u \cong_{\mathsf{sn}} [\![(F:\alpha)^\circ]\!]_u$.

For the proof we use the following observations.

Lemma D.7

i. $(\nu x)(!u(\vec{w}).P \mid !x(\vec{y}).Q) \cong_{\mathsf{sn}} !u(\vec{w}).(\nu x)(P \mid !x(\vec{y}).Q)$ assuming typability.
ii. Assume $\Gamma \vdash P_{1,2} \triangleright A$ with Γ paired and, moreover, $?x_i \in |A|$ $(1 \le i \le n,\ n \ge 0)$. Then $\Gamma \vdash P_1 \cong_{\mathsf{sn}} P_2$ if $P_1|\Pi_i R_i \mapsto^* Q$ implies $P_2|\Pi_i R_i \mapsto^* Q$ and the symmetric case, for any $\Gamma \vdash \Pi_i R_i$ where each R_i is input-prefixed with subject x_i.

PROOF: (i) is immediate by performing the extended reduction at x in both-sides. (ii) is also immediate by $\mapsto \subset \cong_{\mathsf{sn}}$ and from the definition of $\cong_{\mathsf{sn}}$. ■

Using these observations we establish Proposition D.6. We only show the case for $\texttt{let}\ x=zF\ \texttt{in}\ F'$ (other cases are easier). Write $(\ )^\star_u$ for $[\![(\ )^\circ]\!]_u$. Assume $[\![F']\!]_u \stackrel{\text{def}}{=} !u(\vec{w}).P$. Then we have, using induction hypothesis, extended reduction and Lemma D.7 (i),

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

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 we can assume any compensating process at z has this form without loss of generality (by η-expansion and sequentialisation). Immediately $S_1|Q \mapsto^+ (\nu xf)(!u(\vec{v}).P \mid !x(\vec{w}).Q' \mid [\![F]\!]_f) \mid Q$, while

$$\begin{array}{rl}S_2|Q & \mapsto^+ \quad !u(\vec{v}).(\nu fx)([\![F']\!]_f|!x(\vec{w})Q'|P) \mid Q\\ & \cong_{\mathsf{sn}} \quad (\nu fx)(!u(\vec{v}).P \mid !x(\vec{w}).Q' \mid [\![F']\!]_f).\end{array}$$

We can now use Lemma D.7 (ii). ■

D.9. Proof of Proposition 5.8

We prove Proposition 5.8 by establishing a close correspondence between CNFs and their encoding. We use the direct encoding given in D.8

Proposition D.8 *Let $E^\circ \cdot u : T^\circ \vdash P \triangleright !u \to ?A$ be a sequential $\mapsto$-normal form which is η-expanded. Then there is a canonical normal form F such that $[\![F:T]\!]_u \sim P$.*

PROOF: By induction on the size of processes. Let $E^\circ \cdot u : T^\circ \vdash P \triangleright !u \to ?A$ be a sequential $\mapsto$-normal form. Below we write P^x etc. for the prefixed process with x.

i. If $T=\texttt{unit}$, then $P \equiv !u(c)P_0$. If $P_0 \equiv \overline{c}$ then take $F \stackrel{\text{def}}{=} ()$. Otherwise write $P \equiv !u(c)\overline{z}(\vec{w})P'$ with $z \in \texttt{dom}(E)$. If $E(z)=\texttt{unit}$ then $P \equiv !u(c)\overline{z}(w)w.P'$. Using (IH) we can find F' corresponding to $!u(c).P'$, so we let $F \stackrel{\text{def}}{=} \texttt{let}\ ()=z\ \texttt{in}\ F'$. If $E(z)=T_1 \to T_2$ then $P \equiv !u(c)\overline{z}(we)(P'^w|e(x).P''^x)$ With F', F'' corresponding to $P', !u(c).P''$, we let $F \stackrel{\text{def}}{=} \texttt{let}\ x=zF'\ \texttt{in}\ F''$. If $E(z)=T_1\times T_2$ then $P \equiv !u(c)\overline{z}(c)c(xy).P'$. With F' corresponding to $!u(c).P'$, we let $F \stackrel{\text{def}}{=} \texttt{let}\ \langle x,y\rangle =$

z `in` F'. Finally if $E(z) = T_1 + T_2$ then $P \equiv\, !u(c)\overline{z}(w)w[\&_i(x_i).F_i']$. By taking P_i' corresponding to $!u(c).F_i'$, we set $P \equiv$ **`case`** z **`of`** $\{\mathtt{in}_i(x_i).P_i'\}$.

ii. If $T = T_1 \to T_2$, then $P \equiv !u(xz).P_0$. By η-expansion, we can write $P_0 \equiv \overline{z}(c)c(x).P'$. Then by the (IH) there is F' s.t. $[\![F']\!]_x \sim P'$, so we can take $\lambda x.F'$.

iii. If $T = T_1 \times T_2$, then $P \equiv !u(c).P_0$. If $P_0 \equiv \overline{c}(z_1 z_2)\Pi_{i=1,2}R_i^{z_i}$ there are $\{F_i\}$ corresponding to R_i by which we set $F \stackrel{\text{def}}{=} \langle F_1, F_2\rangle$. The rest is similar to (i).

iv. If $T = T_1 + T_2$, then $P \equiv !u(c).P_0$. If $P_0 \equiv \overline{c}\mathtt{in}_1(x_1)P_1^{x_1}$, then take $F \stackrel{\text{def}}{=} \mathtt{in}_1(F_1)$ where F_1 corresponds to P_1. The rest is similar to (i).

Thus we exhausted all cases. ■

E Proofs for Section 6

Lemma E.1 *Let $\Gamma \vdash P \triangleright A \otimes \uparrow x \in \mathsf{NF}_e$ with A closed. Then* (i) *P contains no active linear input channels, and* (ii) *$P \xrightarrow{l}$ 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 is closed. Hence there is no active linear input in P, establishing (i). Since if x is not active it should be under the prefix of a linear input, this also proves (ii), as required. ■

By CR of $\longrightarrow$ and by Theorem 3.1, this establishes Lemma 6.2.

F Syntax and Semantics of the Full Calculus

This section summarises the full calculus (with type annotations on bound names omitted) for reference.

F.1 Syntax

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

Types are generated by the grammars below.

$$
\begin{array}{rclrcl}
\alpha & ::= & \langle \tau, \overline{\tau}\rangle & \tau & ::= & \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \\
\tau_{\mathtt{I}} & ::= & \multicolumn{4}{l}{(\vec{\tau}_{\mathtt{O}})^{\downarrow} \mid (\vec{\tau}_{\mathtt{O}})^{!} \mid [\&_i \vec{\tau}_i]^{\downarrow} \mid [\&_i \vec{\tau}_i]^{!}} \\
\tau_{\mathtt{O}} & ::= & \multicolumn{4}{l}{(\vec{\tau}_{\mathtt{I}})^{\uparrow} \mid (\vec{\tau}_{\mathtt{I}})^{?} \mid [\oplus_i \vec{\tau}_i]^{\uparrow} \mid [\oplus_i \vec{\tau}_i]^{?}}
\end{array}
$$

F.2 Structural Congruence

The *structural congruence* $\equiv$ is the least congruence s.t.:

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

The axioms in the last two clauses are not among usual structural rules for π-calculi.

F.3 Reduction Rules

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

$$
\begin{array}{rl}
(\text{COM}) & x(\vec{y}).P \,|\, \overline{x}(\vec{y})Q \longrightarrow (\nu \vec{y})(P|Q) \\
(\text{BRA}) & x[\&_i(\vec{y}_i).P_i] \,|\, \overline{x}\mathtt{in}_i(\vec{y}_i)Q \longrightarrow (\nu \vec{y}_i)(P_i \,|\, Q) \\
(\text{COM}_!) & !x(\vec{y}).P \,|\, \overline{x}(\vec{y})Q \longrightarrow !x(\vec{y}).P|(\nu \vec{y})(P|Q) \\
(\text{BRA}_!) & !x[\&_i(\vec{y}_i).P_i] \,|\, \overline{x}\mathtt{in}_i(\vec{y}_i)Q \longrightarrow !x[\&_i(\vec{y}_i).P_i]|(\nu \vec{y}_i)(P_i|Q) \\
(\text{RES}) & P \longrightarrow Q \implies (\nu x)P \longrightarrow (\nu x)Q \\
(\text{PAR}) & P \longrightarrow P' \implies P|Q \longrightarrow P'|Q \\
(\text{OUT}) & P \longrightarrow Q \implies \overline{x}(\vec{y})P \longrightarrow \overline{x}(\vec{y})Q \\
(\text{BOUT}) & P \longrightarrow Q \implies \overline{x}\mathtt{in}_i(\vec{y})P \longrightarrow \overline{x}\mathtt{in}_i(\vec{y})Q \\
(\text{CONG}) & P \equiv P' \longrightarrow Q' \equiv Q \implies P \longrightarrow Q
\end{array}
$$

F.4 Transitions

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

$$
\begin{array}{rl}
(\textsc{In}) & x(\vec{y}).P \xrightarrow{x(\vec{y})} P \\
(\textsc{Bra}) & x[\&_{i\in I}(\vec{y}_i).P_i] \xrightarrow{x\mathtt{in}_j(\vec{y}_j)} P_j \\
(\textsc{In}_!) & !x(\vec{y}).P \xrightarrow{x(\vec{y})} !x(\vec{y}).P|P \\
(\textsc{Bra}_!) & !x[\&_{i\in I}(\vec{y}_i).P_i] \xrightarrow{x\mathtt{in}_j(\vec{y}_j)} !x[\&_{i\in I}(\vec{y}_i).P_i]|P_j \\
(\textsc{Out}) & \overline{x}(\vec{z})P \xrightarrow{\overline{x}(\vec{z})} P \\
(\textsc{Sel}) & \overline{x}\mathtt{in}_i(\vec{z})P \xrightarrow{\overline{x}\mathtt{in}_i(\vec{z})} P \\
(\textsc{Com}) & P \xrightarrow{x(\vec{y})} P', Q \xrightarrow{\overline{x}(\vec{y})} Q' \implies P|Q \xrightarrow{\tau} (\nu\vec{y})(P'|Q') \\
(\textsc{Com}_i) & P \xrightarrow{x\mathtt{in}_i(\vec{y})} P', Q \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} Q' \implies P|Q \xrightarrow{\tau} (\nu\vec{y})(P'|Q') \\
(\textsc{Res}) & P \xrightarrow{l} Q, x \notin \mathsf{sbj}(l) \implies (\nu x)P \xrightarrow{l} (\nu x)Q \\
(\textsc{Par}) & P \xrightarrow{l} P', \mathsf{fn}(Q)\cap\mathsf{bn}(l) = \emptyset \implies P|Q \xrightarrow{l} P'|Q \\
(\textsc{Out}\text{-}\xi) & P \xrightarrow{l} P', \mathsf{fn}(l)\cap\{\vec{y}\} = \emptyset \implies \overline{x}(\vec{y})P \xrightarrow{l} \overline{x}(\vec{y})P' \\
(\textsc{Out}\text{-}\xi_i) & P \xrightarrow{l} P', \mathsf{fn}(l)\cap\{\vec{y}_i\} = \emptyset \implies \overline{x}\mathtt{in}_i(\vec{y}_i)P \xrightarrow{l} \overline{x}\mathtt{in}_i(\vec{y}_i)P' \\
(\textsc{Alpha}) & P \equiv_\alpha P' \xrightarrow{l} Q' \equiv_\alpha Q \implies P \xrightarrow{l} Q
\end{array}
$$