Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Extensional and Non-extensional Functions as Processes

Extensional and Non-extensional Functions as Processes

Ken Sakayori
University of Bologna and Inria

Davide Sangiorgi
University of Bologna and Inria

Abstract—**Following Milner's seminal paper, the representation of functions as processes has received considerable attention. For pure λ-calculus, the process representations yield (at best) *non-extensional* λ-theories (i.e., β rule holds, whereas η does not).**

In the paper, we study how to obtain *extensional* representations, and how to move between extensional and non-extensional representations. Using Internal π, Iπ (a subset of the π-calculus in which all outputs are bound), we develop a refinement of Milner's original encoding of functions as processes that is *parametric* on certain abstract components called *wires*. These are, intuitively, processes whose task is to connect two end-point channels. We show that when a few algebraic properties of wires hold, the encoding yields a λ-theory. Exploiting the symmetries and dualities of Iπ, we isolate three main classes of wires. The first two have a sequential behaviour and are dual of each other; the third has a parallel behaviour and is the dual of itself. We show the adoption of the parallel wires yields an extensional λ-theory; in fact, it yields an equality that coincides with that of Böhm trees with infinite η. In contrast, the other two classes of wires yield non-extensional λ-theories whose equalities are those of the Lévy-Longo and Böhm trees.

I. Introduction

Milner's work [1], [2] on the encoding of the pure λ-calculus into the π-calculus is generally considered a landmark paper in the area of semantics and programming languages. The encoding of the λ-calculus is a significant test of expressiveness for the π-calculus. The encoding also gives an interactive semantics to the λ-calculus, which allows one to analyse it using the instruments available in the π-calculus. After Milner's seminal work, a number of encoding variants have been put forward (e.g. [3] and references therein) by modifying the target language (often to a subcalculus of the π-calculus) or the encoding itself. The correctness of these encodings is usually supported by the operational correspondence against a certain evaluation strategy of the λ-calculus and by the validity of the β-rule, $(\lambda x. M)N = M\{N/x\}$. (In this paper, by validity of a λ-calculus rule with respect to a certain process encoding $\{\!\!\{\,\cdot\,\}\!\!\}$, we mean that $\{\!\!\{M\}\!\!\} \approx \{\!\!\{N\}\!\!\}$ for all instances $M = N$ of (the congruence closure of) the rule, where $\approx$ is a basic behavioural equivalence for the pure processes, such as ordinary bisimilarity.)

The equality on λ-terms induced by the encoding has also been investigated; in this equality two λ-terms M and N are equal when their images are behaviourally equivalent processes. For Milner's original (call-by-name) encoding, such an equality coincides with the Lévy-Longo tree (LT) equality [4], [5] (the result is by large independent of the behavioural equivalence adopted for the processes [6]). It has also been shown how to recover the Böhm tree (BT) equality [3], by modifying Milner's encoding — allowing reductions underneath a λ-abstraction — and selecting divergence-sensitive behavioural equivalences on processes such as must-testing.

Tree structures play a pivotal role in the λ-calculus. For instance, trees allow one to unveil the computational content hidden in a λ-term, with respect to some relevant minimal information. In BTs the information is the head normal forms, whereas in LTs it is the weak head normal forms. BT and LT equalities coincide with the local structures of well-known models of the λ-calculus, such as Plotkin and Scott's P_ω [7], [8], and the *free lazy Plotkin-Scott-Engeler models* [9], [10], [11].

In BTs and LTs, the computational content of a λ-term is unveiled using the β-rule alone. Such structures are sometimes called *non-extensional*, as opposed to the *extensional* structures, in which the β-rule is coupled with the η-rule, $M = \lambda x. M x$ (for x not free in M). In extensional theories two functions are equated if, whenever applied to the same argument, they yield equal results. A well-known extensional tree-structure are BTs with infinite η, shortly $\mathrm{BT}_{\eta\infty}$s. The equality of $\mathrm{BT}_{\eta\infty}$s coincides with that of Scott's D_∞ model [8], historically the first model of the untyped λ-calculus. A seminal result by Wadsworth [12] shows that the $\mathrm{BT}_{\eta\infty}$s are intimately related to the head normal forms, as the $\mathrm{BT}_{\eta\infty}$ equality coincides with contextual equivalence in which the head normal forms are the observables.

In representations of functions as processes, extensionality and the η-rule, even in their most basic form, have always appeared out of reach. For instance, in Milner's encoding, x and $\lambda y. x y$ have quite different behaviours: the former process is a single output particle, whereas the latter process has an infinite behaviour and, moreover, the initial action is an input.

The general goal of this paper is to study extensionality in the representation of functions as processes. In partic-

Encoding	Parameter (wires)	Characterises
$\mathcal{A}_{\texttt{IO}}$	I-O wires	BT
$\mathcal{A}_{\texttt{P}}$	P wires	$\text{BT}_{\eta\infty}$
$\mathcal{A}_{\texttt{OI}}$	O-I wires	LT

ular, we wish to understand if and how one can derive extensional representations, and the difference between extensional and non-extensional representations from a process perspective.

We outline the main technical contributions. We develop a refinement of Milner's original encoding of functions, using Internal π (Iπ), a subcalculus of the π-calculus in which only bound names may be exported. The encoding makes use of certain abstract components called *wires*. These are, intuitively, processes whose task is to connect two end-point channels; and when one of the two end-points is restricted, the wires behave as substitutions. In the encoding, wires are called 'abstract' because their definitions are not made explicit. We show that assuming a few basic algebraic properties of wires (having to do with transitivity of wires and substitution) is sufficient to obtain a λ-theory, i.e. the validity of the β-rule.

We then delve into the impact of the concrete definition of the wires, notably on the equivalence on λ-terms induced by the encoding. In the π-calculus literature, the most common form of wire between two channels a and b is written $!a(u).\overline{b}\langle u \rangle$ (or $a(u).\overline{b}\langle u \rangle$, if only needed once), and sometimes called a *forwarder* [13], [14]. In Iπ, free outputs are forbidden and such a wire becomes a recursively-defined process. We call this kind of wires *I-O wires*, because of their 'input before output' behaviour. Exploiting the properties of Iπ, e.g., its symmetries and dualities, we identify two other main kinds of wires: the *O-I wires*, with an 'output before input' behaviour and which are thus the dual of the I-O wires; and the *P wires*, or *parallel wires*, where input and output can fire concurrently (hence such wires are behaviourally the same as their dual).

We show that moving among these three kinds of wire corresponds to moving among the three above-mentioned tree structures of the λ-calculus, namely BTs, LTs, $\text{BT}_{\eta\infty}$s. Precisely, we obtain BTs when adopting the ordinary I-O wires; LTs when adopting the O-I wires; and $\text{BT}_{\eta\infty}$s when adopting the P wires. This also implies that P wires allow us to validate the η-rule (in fact both η and infinite η). The results are summarised in Table I, where $\mathcal{A}_{\texttt{X}}$ is the concrete encoding in which the X wires are used.

We are not aware of results in the literature that produce an *extensional* λ-theory from a processes model, let alone that derive the $\text{BT}_{\eta\infty}$ equality. We should also stress that the choice of the wire is the *only* modification needed for switching among the three tree structures: the encoding of the λ-calculus is otherwise the same, nor does it change

the underlying calculus and its behavioural equivalence (namely, Iπ and bisimilarity).

There are various reasons for using Iπ in our study. The first and most important reason has to do with the symmetries and dualities of Iπ, as hinted above. The second reason is proof techniques: in the paper we use a wealth of proof techniques, ranging from algebraic laws to forms of 'up-to bisimulation' and to unique solutions of equations; not all of them are available in the ordinary π-calculus. The third reason has to do with η-rule. In studies of the expressiveness of Iπ in the literature [15] the encoding of the free-output construct into Iπ resembles an (infinite) η-expansion. The essence of the encoding is the following transformation (which needs to be recursively applied to eliminate all free outputs):

$$\overline{a}\langle p \rangle \mapsto \boldsymbol{\nu} q \ (\overline{a}\langle q \rangle \mid q(\widetilde{y}).\overline{p}\langle\widetilde{y}\rangle). \tag{1}$$

A free output of p is replaced by a bound output, that is, an output of a freshly created name q (for simplicity, we assume that p is meant to be used only once by the recipient). The transformation requires *localised* calculi [16], in which the recipient of a name may only use it in output, and resembles an η-expansion of a variable of the λ-calculus in that, intuitively, direct access to the name p is replaced by access to the function $\lambda\widetilde{y}.\overline{p}\langle\widetilde{y}\rangle$.

A possible connection between Iπ and η-expansion may also be found in papers such as [17], where η-expanded proofs (proofs in which the identity rule is only applied to atomic formulas) are related to (session-typed) processes with bound outputs only. Yet, the technical link with our works appears weak because the wires that we use to achieve extensionality (the P wires of Table I) are behaviourally quite different from the process structures mentioned above.

We derive the encoding into Iπ in two steps. The first step consists, intuitively, in transplanting Milner's encoding into Iπ, by replacing free outputs with bound outputs plus wires, following the idea in (1) above. However, (1) is only valid in localised calculi, whereas Milner's encoding also requires the *input* capability of names to be transmitted. Therefore we have to modify the wire in (1), essentially inverting the two names p and q. The correctness of the resulting transformation relies on properties about the usage of names that are specific to the representation of functions. The second step adopted to derive the encoding consists of allowing reductions underneath a λ abstraction; that is, implementing a *strong* reduction strategy. This transformation is necessary in order to mimic the computation required to obtain head normal forms.

Encodings of strong reduction strategies have appeared in the literature; they rely on the possibility of encoding non-blocking prefixes (sometimes called *delayed* in the literature) [18], [19], [20], [21], [14], [16], [3], i.e., prefixes $\mu : P$ in which actions from P may fire before μ, as long as μ does not bind names of the action. The encodings of

non-blocking prefixes in the literature require the names bound in μ to be localised. Here again, the difficulty was to adapt the schema to non-localised names. Similar issues arise within wires, as their definition also requires certain prefixes to be non-blocking.

Structure of the paper Section II recalls background material on λ-calculus and Iπ. In Section III, we introduce wires and permeable prefixes. In Section IV, we present the abstract encoding, using the abstract wires, and the assumptions we make on wires; we then verify that such assumptions are sufficient to obtain a λ-theory. Section V defines an optimised abstract encoding, which will be useful for later proofs. In Section VI, we introduce the three classes of concrete wires, and show that they satisfy the required assumptions for wires. In Section VII, we pick the I-O wires and O-I wires, and prove full abstraction for LTs and BTs. In Section VIII, we do the same for the P wires and $BT_{\eta\infty}$s. Section IX discusses further related work and possible future developments. For lack of space, proofs of the main results are only sketched. The reader may find the details in the full version [22].

II. BACKGROUND

A tilde represents a tuple. The i-th element of a tuple $\widetilde{P}$ is referred to as P_i. All notations are extended to tuples componentwise.

A. The λ-calculus

We let x and y range over the set of λ-calculus variables. The set Λ of λ-terms is defined by the grammar

$$M ::= x \mid \lambda x.\, M \mid M_1\, M_2\,.$$

Free variables, closed terms, substitution, α-conversion etc. are defined as usual [23]; the set of free variables of M is $\mathsf{fv}(M)$. Here and in the rest of the paper (including when reasoning about processes), we adopt the usual 'Barendregt convention'. This will allow us to assume freshness of bound variables and names whenever needed. We group brackets on the left; therefore MNL is $(MN)L$. We abbreviate $\lambda x_1.\cdots.\lambda x_n.\, M$ as $\lambda x_1 \cdots x_n.\, M$, or $\lambda \widetilde{x}.\, M$. Symbol Ω stands for the always-divergent term $(\lambda x.\, xx)(\lambda x.\, xx)$.

A number of reduction relations are mentioned in this paper. The (standard) β-*reduction* relation $M \to N$ is the relation on λ-terms induced by the following rules:

$$[\beta]\ \frac{}{(\lambda x.\, M)\, N \to M\{N/x\}} \qquad [\mu]\ \frac{N \to N'}{M\, N \to M\, N'}$$

$$[\nu]\ \frac{M \to M'}{M\, N \to M'\, N} \qquad [\xi]\ \frac{M \to M'}{\lambda x.\, M \to \lambda x.\, M'}$$

The (weak) *call-by-name reduction* relation uses only the β and ν rules, whereas *strong call-by-name*, written $\to_{\mathsf{sn}}$, also has ξ; the *head reduction*, written $\to_{\mathsf{h}}$, is a deterministic variant of $\to_{\mathsf{sn}}$ in which the redex contracted is the head one, i.e., $(\lambda y.\, M_0)\, M_1$ of $\lambda \widetilde{x}.\, (\lambda y.\, M_0)\, M_1 \cdots M_n$. *Head normal forms* are of the form $\lambda \widetilde{x}.\, y\, \widetilde{M}$. As usual, we use a double arrow to indicate the reflexive and transitive closure of a reduction relation, as in $\Longrightarrow$ and $\Longrightarrow_{\mathsf{h}}$. A term

M *has a head normal form* N if $M \Longrightarrow_{\mathsf{h}} N$ and N is the (unique) head normal form. Terms that do not have a head normal form are called *unsolvable*. An unsolvable M has an *order of unsolvability* n, if n is the largest natural number such that $M \Longrightarrow_{\mathsf{h}} \lambda x_1 \ldots x_n.\, M'$, for $n \geq 0$, and some $x_1, \ldots, x_n, M'$. If there is no such largest number, then M is *of order* ω. For instance, Ω is an unsolvable of order 0 and $\lambda x.\, \Omega$ of order 1.

We recall the definitions of Lévy-Longo trees and Böhm trees; then in the equality induced by such trees two terms are related if their trees are the same (as usual modulo α-conversion). In contrast, we present the definition of Böhm tree equality up-to infinite η-expansion as a bisimilarity, as the proofs exploit this bisimulation-based definition.

Definition II.1 (Lévy-Longo trees and Böhm trees)**.** The *Lévy-Longo tree* of M is the labelled tree, $\mathrm{LT}(M)$, defined coinductively as follows:

1) $\mathrm{LT}(M) = \top$ if M is an unsolvable of order ω;
2) $\mathrm{LT}(M) = \lambda x_1 \ldots x_n.\bot$ if M is an unsolvable of order $n < \omega$;
3) $\mathrm{LT}(M)$ is the tree with $\lambda \widetilde{x}.\, y$ as the root and $\mathrm{LT}(M_1) \ldots \mathrm{LT}(M_n)$ as the children, if M has head normal form $\lambda \widetilde{x}.\, y\, M_1 \cdots M_n$ with $n \geq 0$.

The definition of Böhm trees (BTs) is obtained from that of LTs using BT in place of LT, and demanding that $\mathrm{BT}(M) = \bot$ whenever M is unsolvable (in place of clauses (1) and (2)).

An η-expansion of a BT, whose root is $\lambda \widetilde{x}.\, y$ and children are $\mathrm{BT}(M_1), \ldots \mathrm{BT}(M_n)$, is given by a tree whose root is $\lambda \widetilde{x}z.\, y$ and children are $\mathrm{BT}(M_1), \ldots \mathrm{BT}(M_n), z$. Informally, an infinite η-expansion of a BT is obtained by allowing this expansion at each step of the clause (3). Equality over such trees can be formalised as a bisimilarity in which (finite) η-expansion is allowed at each step of the bisimulation game.

Definition II.2 ([24])**.** A symmetric relation $\mathcal{R}$ on λ-terms is a $BT_{\eta\infty}$-*bisimulation* if, whenever $M\, \mathcal{R}\, N$, either one of the following holds:

1) M and N are unsolvable
2) $M \Longrightarrow_{\mathsf{h}} \lambda x_1 \ldots x_{l+m}.\, y\, M_1 \cdots M_{n+m}$ and $N \Longrightarrow_{\mathsf{h}} \lambda x_1 \ldots x_l.\, y\, N_1 \cdots N_n$, where $x_{l+1}, \ldots, x_{l+m}$ are not free in $y\, N_1, \ldots, N_n$, and $M_i\, \mathcal{R}\, N_i$ for $1 \leq i \leq n$, and also $M_{n+j}\, \mathcal{R}\, x_{l+j}$ for $1 \leq j \leq m$
3) the symmetric case, where N reduces to a head normal form with more leading λs.

The largest $BT_{\eta\infty}$-bisimulation is called $BT_{\eta\infty}$-*bisimilarity*. We also write $\mathrm{BT}_{\eta\infty}(M) = \mathrm{BT}_{\eta\infty}(N)$ when M and N are $BT_{\eta\infty}$-bisimilar.

Example II.1. We have $\mathrm{LT}(\lambda x.\, \Omega) = \lambda x.\bot$, whereas $\mathrm{BT}(\lambda x.\, \Omega) = \bot$. For $\Xi \stackrel{\text{def}}{=} (\lambda xy.\, x\, x)(\lambda xy.\, x\, x)$ we have $\mathrm{LT}(\Xi) = \top$, whereas $\mathrm{BT}(\Xi) = \bot$.

Example II.2. Let J be a term such that $J\, z \Longrightarrow_{\mathsf{h}} \lambda y.\, z\, (J\, y)$, which is easy to define using a fixed-point

combinator. Intuitively, the term $J\,z$ can be considered as the 'limit of the sequence of η-expansions'

$$z \to_\eta \lambda z_1.\, z\, z_1 \to_\eta \lambda z_1.\, z\, (\lambda z_2.\, z_1\, z_2) \to_\eta \cdots$$

The terms z and $J\,z$ have different Böhm trees, as $\mathrm{BT}(z) = z$ whereas $\mathrm{BT}(J\,z)$ has infinitely many nodes, the root being $\lambda z_1.\, z$. However, $\mathrm{BT}_{\eta\infty}(J\,z) = \mathrm{BT}_{\eta\infty}(z)$ as the two terms can be equated using an infinite form of η-expansion.

B. Internal π-calculus

In all encodings we consider, the encoding of a λ-term is parametric on a name, i.e., it is a function from names to π-calculus processes. We also need parametric processes (over one or several names) for writing recursive process definitions and equations. We call such parametric processes *abstractions*. The actual instantiation of the parameters of an abstraction F is done via the *application* construct $F\langle\widetilde{a}\rangle$. Processes and abstractions form the set of *agents*. Small letters $a, b, \ldots, x, y, \ldots$ range over the infinite set of names. The grammar of Iπ is thus:

$$
\begin{array}{rcll}
A & ::= & P \;\mid\; F & \text{(agents)} \\
P & ::= & \mathbf{0} \;\mid\; a(\widetilde{b}).\,P \;\mid\; \overline{a}(\widetilde{b}).\,P \;\mid\; \boldsymbol{\nu} a\,P & \text{(processes)} \\
  &   & \mid\; P_1 \mid P_2 \;\mid\; !a(\widetilde{b}).\,P \;\mid\; F\langle\widetilde{a}\rangle & \\
F & ::= & (\widetilde{a})\,P \;\mid\; \mathsf{K} & \text{(abstractions)}
\end{array}
$$

The operators used have the usual meanings. In prefixes $a(\widetilde{b})$ and $\overline{a}(\widetilde{b})$, we call a the *subject*. We often abbreviate $\boldsymbol{\nu} a\,\boldsymbol{\nu} b\,P$ as $(\boldsymbol{\nu} a, b)\,P$. Prefixes, restriction, and abstraction are binders and give rise in the expected way to the definition of *free names*, *bound names*, and *names* of an agent, respectively indicated with $\mathsf{fn}(-)$, $\mathsf{bn}(-)$, and $\mathsf{n}(-)$, as well as that of α-conversion. An agent is *name-closed* if it does not contain free names. In the grammar, K is a *constant*, used to write recursive definitions. Each constant K has a defining equation of the form $\mathsf{K} \stackrel{\mathrm{def}}{=} (\widetilde{x})\,P$, where $(\widetilde{x})\,P$ is name-closed; $\widetilde{x}$ are the formal parameters of the constant (replaced by the actual parameters whenever the constant is used). Replication could be avoided in the syntax since it can be encoded with recursion. However its semantics is simple, and it is a useful construct for encodings; thus we chose to include it in the grammar.

Since the calculus is polyadic, we assume a *sorting system* [25] to avoid disagreements in the arities of the tuples of names carried by a given name and in applications of abstractions. In Milner's encoding (written in the polyadic π-calculus) as well as in all encodings in the paper, there are only two sorts of names: *location names*, and *variable names*. Location names carry a pair of a variable name and a location name; variable names carry a single location name. Using $p, q, r, \ldots$ for location names, and $x, y, z, \ldots$ for variable names, the forms of the possible prefixes are:

$$p(x, q).\,P \;\mid\; x(p).\,P \;\mid\; \overline{p}(x, q).\,P \;\mid\; \overline{x}(p).\,P$$

This sorting will be maintained throughout the paper. Hence process transformations and algebraic laws will be given with reference to such a sorting.

The operational semantics of Iπ is standard [3] (see [22]). The reference behavioural equivalence for Iπ is (weak) bisimilarity; it coincides with barbed congruence, assuming image-finiteness. We also use the *expansion* preorder, written $\lesssim$, an asymmetric variant of $\approx$ in which, intuitively, $P \lesssim Q$ holds if $P \approx Q$ but also Q has at least as many τ-moves as P. As usual, $\stackrel{\mu}{\Longrightarrow}$ is $\Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$, and $\stackrel{\hat{\mu}}{\Longrightarrow}$ is $\stackrel{\mu}{\Longrightarrow}$ for $\mu \neq \tau$ and $\Longrightarrow$ otherwise.

Definition II.3 (Bisimilarity and Expansion)**.** A symmetric relation $\mathcal{R}$ on Iπ-processes is a *bisimulation*, if whenever $P \,\mathcal{R}\, Q$ and $P \stackrel{\mu}{\longrightarrow} P'$, then $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ for some Q' with $P' \,\mathcal{R}\, Q'$. Processes P and Q are *bisimilar*, written $P \approx Q$, if $P \,\mathcal{R}\, Q$ for some bisimulation $\mathcal{R}$.

A relation $\mathcal{R}$ over processes is an *expansion* if $P \,\mathcal{R}\, Q$ implies, whenever $P \stackrel{\mu}{\longrightarrow} P'$ (resp. $Q \stackrel{\mu}{\longrightarrow} Q'$), there exists Q' (resp. P') such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ (resp. $P \stackrel{\widehat{\mu}}{\longrightarrow} P'$) and $P' \,\mathcal{R}\, Q'$. Here, $\stackrel{\widehat{\mu}}{\longrightarrow}$ is $\stackrel{\mu}{\longrightarrow}$ if $\mu \neq \tau$ and is $=$ or $\stackrel{\tau}{\longrightarrow}$ if $\mu = \tau$. We say that Q *expands* P, written $P \lesssim Q$, if $P \,\mathcal{R}\, Q$, for some expansion $\mathcal{R}$.

All behavioural relations are extended to abstractions by requiring ground instantiation of the parameters.

1) Proof techniques: In the proofs, we often use well-known algebraic laws, notably laws for private replications, and up-techniques for bisimilarity, notably bisimulations up-to expansion and contexts; we use up-to expansion rather than up-to weak bisimulation as the latter is known to be unsound. Again, we refer to [22] for details.

We briefly recall the 'unique solution of equations' technique [26]. *Equation variables* X, Y, Z are used to write equations. The body of an equation is a name-closed abstraction possibly containing equation variables (that is, applications can also be of the form $X\langle\widetilde{a}\rangle$). We use E to range over expression bodies; and $\mathcal{E}$ to range over systems of equations, defined as follows. In all the definitions, the indexing set I can be infinite.

Definition II.4. Assume that, for each i of a countable indexing set I, we have a variable X_i, and an expression E_i, possibly containing variables. Then $\{X_i = E_i\}_{i \in I}$ (sometimes written $\widetilde{X} = \widetilde{E}$) is a *system of equations*. (There is one equation for each variable X_i.) A system of equations is *guarded* if each occurrence of a variable in the body of an equation is underneath a prefix.

We write $E[\widetilde{F}]$ for the abstraction obtained by replacing in E each occurrence of the variable X_i with the abstraction F_i. This is a syntactic replacement, with instantiation of the parameters: e.g., replacing X with $(\widetilde{x})P$ in $X\langle\widetilde{a}\rangle$ amounts to replacing $X\langle\widetilde{a}\rangle$ with $P\{\widetilde{a}/\widetilde{x}\}$.

Definition II.5. Suppose $\{X_i = E_i\}_{i \in I}$ is a system of equations. We say that:

- $\widetilde{F}$ is a *solution of the system of equations for* $\approx$ if for each i it holds that $F_i \approx E_i[\widetilde{F}]$.
- The system has *a unique solution for* $\approx$ if whenever $\widetilde{F}$ and $\widetilde{G}$ are both solutions for $\approx$, we have $\widetilde{F} \approx \widetilde{G}$.

Definition II.6 (Syntactic solutions)**.** The *syntactic solutions* of a system of equations $\{X_i = E_i\}_{i \in I}$ are the recursively defined constants $\mathsf{K}_{\widetilde{E},i} \overset{\text{def}}{=} E_i[\widetilde{\mathsf{K}_{\widetilde{E}}}]$, for $i \in I$.

The syntactic solutions of a system of equations are indeed solutions of it. The unique-solution technique relies on an analysis of divergences. A process P *diverges* if it can perform an infinite sequence of internal moves, possibly after some visible ones (i.e., actions different from τ). Formally, this holds if there are processes P_i, $i \geq 0$, and some n such that $P = P_0 \xrightarrow{\mu_0} P_1 \xrightarrow{\mu_1} P_2 \xrightarrow{\mu_2} \dots$ and for all $i > n$, $\mu_i = \tau$. We call a *divergence of P* the sequence of transitions $\left(P_i \xrightarrow{\mu_i} P_{i+1}\right)_{i \geq 0}$. An abstraction F has a divergence if the process $F\langle \widetilde{a} \rangle$ has a divergence, where $\widetilde{a}$ are fresh names.

Theorem II.1 ([27])**.** A guarded system of equations whose syntactic solutions are agents with no divergences has a unique solution for $\approx$.

III. Wires and Permeable Prefixes

We introduce the abstract notion of *wire* process; and, as a syntactic sugar, the process constructs for *permeable prefixes*. Wires and permeable prefixes will play a central role in the technical development in the following sections. We use the notation $a \leftrightarrow \bar{b}$ for an *abstract wire*; this is, intuitively, a special process whose purpose is to connect the output-end of a with the input-end of b (thus $a \leftrightarrow \bar{b}$ itself will use a in input and b in output). We call such wires 'abstract' because we will not give a definition for them. We only state (Section IV) some behavioural properties that are expected to hold, and that have mainly to do with substitutions; approximately:

1) if P uses b only in input, then
   $\boldsymbol{\nu} b \, (a \leftrightarrow \bar{b} \mid P) \gtrsim P\{a/b\}$
2) dually, if P uses a only in output, then
   $\boldsymbol{\nu} a \, (a \leftrightarrow \bar{b} \mid P) \gtrsim P\{b/a\}$

Further conditions will however be needed on P for such properties to hold (e.g., in (1), the input at b in P should be 'at the top-level', and in (2), the outputs at a in P should be 'asynchronous'.) Special cases of (1) and (2) are forms of transitivity for wires, with the common name restricted:

3) $\boldsymbol{\nu} b \, (a \leftrightarrow \bar{b} \mid b \leftrightarrow \bar{c}) \gtrsim (b \leftrightarrow \bar{c})\{a/b\} = (a \leftrightarrow \bar{b})\{c/b\} = a \leftrightarrow \bar{c}$.

When (1) holds we say that P is *I-respectful with respect to $a \leftrightarrow \bar{b}$*; similarly when (2) holds we say that P is *O-respectful with respect to $a \leftrightarrow \bar{b}$*. When (3) holds, for any a, b, c of the same sort, we say that *wires are transitive*.

As we have two sorts of names in the paper (location names and variable names), we will correspondingly deal with two sorts of wires, *location wires* and *variable wires*. In fact, location wires will be the key structures: definitions and properties for the variable wires will be adjusted accordingly, so to guarantee the desired properties of the location wires.

We write $a(\widetilde{b})\colon P$ and $\bar{a}(\widetilde{b})\colon P$ for a *permeable input* and a *permeable output*. Intuitively, a permeable prefix only blocks actions involving the bound names of the prefix. For instance, a permeable input $a(\widetilde{b})\colon P$, as an ordinary input, is capable of producing action $a(\widetilde{b})$ thus yielding the derivative P. However, in contrast with the ordinary input, in $a(\widetilde{b})\colon P$ the process P is active and running, and can thus interact with the outside environment; the only constraint is that the actions from P involving the bound names $\widetilde{b}$ cannot fire for as long as the top prefix $a(\widetilde{b})$ is not consumed.

Given the two sorts of names that will be used in the paper, the possible forms of permeable prefixes are:

$$p(x, q)\colon P \quad \big| \quad \bar{p}(x, q)\colon P \quad \big| \quad x(p)\colon P \quad \big| \quad \bar{x}(p)\colon P$$

Moreover, it will always be the case that in a prefix $p(x, q)\colon P$ process P uses x only in output and q only once in input, and conversely for $\bar{p}(x, q)\colon P$; and in $x(p)\colon P$ process P uses p only once in input, and conversely for $\bar{x}(p)\colon P$.

We stress that permeable prefixes should be taken as syntactic sugar; formally they are defined from ordinary prefixes and wires as follows

$$p(x, q)\colon P \overset{\text{def}}{=} (\boldsymbol{\nu} x, q) \; (p(x', q').\, (x \leftrightarrow \bar{x}' \mid q' \leftrightarrow \bar{q}) \mid P)$$

$$x(p)\colon P \overset{\text{def}}{=} \boldsymbol{\nu} p \; (x(p').\, p' \leftrightarrow \bar{p} \mid P)$$

$$\bar{p}(x, q)\colon P \overset{\text{def}}{=} (\boldsymbol{\nu} x, q) \; (\bar{p}(x', q').\, (x' \leftrightarrow \bar{x} \mid q \leftrightarrow \bar{q}') \mid P)$$

$$\bar{x}(p)\colon P \overset{\text{def}}{=} \boldsymbol{\nu} p \; (\bar{x}(p').\, p \leftrightarrow \bar{p}' \mid P)$$

Such definitions thus depend on the concrete forms of wires adopted. The definitions behave as intended only when the processes underneath the permeable prefixes are respectful. For example, we have

$$p(x, q)\colon P \xrightarrow{p(x, q)}$$
$$(\boldsymbol{\nu} x', q') \; \left(x' \leftrightarrow \bar{x} \mid q \leftrightarrow \bar{q}' \mid P\{x', q'/x, q\}\right) \gtrsim P$$

if P is I-respectful with respect to $q' \leftrightarrow \bar{q}$ and O-respectful with respect to $x \leftrightarrow \bar{x}'$, for fresh q' and x'.

Later, when the abstract wires will be instantiated to concrete wires, we will study properties of I-respectfulness and O-respectfulness, as well as, correspondingly, properties of permeable prefixes, in the setting of encodings of functions.

IV. Abstract Encoding

This section introduces the abstract encoding of λ-terms into Iπ-processes. We call the encoding 'abstract' because

$$\mathcal{A}[\![x]\!]_p \overset{\text{def}}{=} \bar{x}(p') \colon p \leftrightarrow \bar{p}'$$

$$\mathcal{A}[\![\lambda x.\, M]\!]_p \overset{\text{def}}{=} p(x,q) \colon \mathcal{A}[\![M]\!]_q$$

$$\mathcal{A}[\![M\,N]\!]_p \overset{\text{def}}{=} \boldsymbol{\nu}q\; (\; \mathcal{A}[\![M]\!]_q \mid$$
$$\bar{q}(x,p') \colon (!x(r).\,\mathcal{A}[\![N]\!]_r \mid p \leftrightarrow \bar{p}')\;)$$

Fig. 1. The abstract encoding $\mathcal{A}$.

it uses the abstract wires discussed in the previous section. In other words, the encoding is *parametric* with respect to the concrete definition of the wires. We then prove that, whenever the wires satisfy a few basic laws, the encoding yields a λ-theory.

A. Definition of the abstract encoding

We begin by recalling Milner's original encoding $\mathcal{M}$ of (call-by-name) λ-calculus into the π-calculus [2], [25]:

$$\mathcal{M}[\![x]\!]_p \overset{\text{def}}{=} \bar{x}\langle p \rangle$$

$$\mathcal{M}[\![\lambda x.\, M]\!]_p \overset{\text{def}}{=} p(x,q).\,\mathcal{M}[\![M]\!]_q$$

$$\mathcal{M}[\![M\,N]\!]_p \overset{\text{def}}{=} (\boldsymbol{\nu}q,x)\; (\mathcal{M}[\![M]\!]_q \mid \bar{q}\langle x,p\rangle \mid !x(r).\,\mathcal{M}[\![N]\!]_r)$$

The encoding of a λ-term M is parametric over a port p, which can be thought of as the *location* of M, for p represents the unique port along which M may be called by its environment, thus receiving two names: (a trigger for) its argument and the location to be used for the next interaction. Hence, $\mathcal{M}$ (as well as all the encodings in the paper) is a function from λ-terms to abstractions of the form $(p)\,P$. We write $\mathcal{M}[\![M]\!]_p$ as a shorthand for $\mathcal{M}[\![M]\!]\langle p \rangle$. A function application of the λ-calculus becomes, in the π-calculus, a particular form of parallel combination of two agents, the function and its argument. An argument of an application is translated as a replicated server, that can be used as many times as needed, each time providing a name to be used as location for the following computation.

In Figure 1 we report the abstract encoding $\mathcal{A}$. There are two main modifications from Milner's encoding $\mathcal{M}$:

1) The encoding uses $I\pi$, rather than π-calculus; for this, all free outputs are replaced by a combination of bound outputs and wires.
2) A permeable input is used, in place of an ordinary input, in the translation of abstraction so to allow reductions underneath a λ-abstraction. (We thus implement a *strong* call-by-name strategy.)

We report a few basic conditions that will be required on wires. The main ones concern the behaviour of wires as substitutions and transitivity of wires.

Definition IV.1 (Wires)**.** As a convention, we assume that names a, b, c are of the same sort, either location names or variable names. *Wires* $a \leftrightarrow \bar{b}$ are processes that satisfy the following properties:

1) The free names of $a \leftrightarrow \bar{b}$ are a and b. Furthermore, $a \leftrightarrow \bar{b}$ only uses a in input and b in output.
2) If $a \leftrightarrow \bar{b} \xrightarrow{\mu} P$ for some P, then $\mu \neq \tau$.
3) $\boldsymbol{\nu}b\,(a \leftrightarrow \bar{b} \mid b \leftrightarrow \bar{c}) \gtrsim a \leftrightarrow \bar{c}$.
4) $\boldsymbol{\nu}q\,(p \leftrightarrow \bar{q} \mid q(x,r)\colon P) \gtrsim p(x,r)\colon P$, provided that $(\boldsymbol{\nu}x,r\,)(x \leftrightarrow \bar{x}' \mid r' \leftrightarrow \bar{r} \mid P) \gtrsim P\{x',r'\!/x,r\}$, where x', r' are fresh names.
5) $\boldsymbol{\nu}p\,(p \leftrightarrow \bar{q} \mid \bar{p}(x,r)\colon P) \gtrsim \bar{q}(x,r) \colon P$, provided that $(\boldsymbol{\nu}x,r\,)(x' \leftrightarrow \bar{x} \mid r \leftrightarrow \bar{r}' \mid P) \gtrsim P\{x',r'\!/x,r\}$, where x', r' are fresh names.
6) $\boldsymbol{\nu}y\,(x \leftrightarrow \bar{y} \mid !y(p).\,P) \gtrsim !x(p).\,P$, provided that $y \notin \mathsf{fn}(P)$ and $\boldsymbol{\nu}p(p' \leftrightarrow \bar{p} \mid P) \gtrsim P\{p'\!/p\}$, where p' is fresh.

7) $\boldsymbol{\nu}x\,(x \leftrightarrow \bar{y} \mid \bar{x}(p) \colon P) \gtrsim \bar{y}(p) \colon P$, provided that $x \notin \mathsf{fn}(P)$ and $\boldsymbol{\nu}p\,(p \leftrightarrow \bar{p}' \mid P) \gtrsim P\{p'\!/p\}$, where p' is fresh.
8) $x \leftrightarrow \bar{y}$ is a replicated input process at x, i.e. $x \leftrightarrow \bar{y} = !x(p).\,P$ for some P.

Condition 1 is a simple syntactic requirement. Condition 2 says that wires are 'optimised' in that they cannot do any immediate internal interaction (this requirement, while not mandatory, facilitates a few proofs). Law 3 is about the transitivity of wires. Laws 4-7 show that wires act as substitutions for permeable inputs, permeable outputs and replicated input prefixes. We do not require similar laws for ordinary prefixes, e.g., as in

$$\boldsymbol{\nu}p\,(p \leftrightarrow \bar{q} \mid \bar{p}(x,r).\,P) \gtrsim \bar{q}(x,r).\,P$$

because wires break the strict sequentiality imposed by such prefixes (essentially transforming an ordinary prefix into a permeable one: only for the process on the right any action from P is blocked until the environment accepts an interaction at q). Condition 8 requires $x \leftrightarrow \bar{y}$ to be an input replicated processes, and is useful so to be able to use the replication laws.

Hereafter we assume that $p \leftrightarrow \bar{q}$ and $x \leftrightarrow \bar{y}$ are indeed wires, i.e., processes that satisfy the requirements of Definition IV.1. We can therefore exploit such requirements to derive properties of the abstract encoding.

Lemma IV.1 shows that the processes encoding functions are I-respectful with respect to the location wires, and O-respectful with respect to the variable wires.

Lemma IV.1.

1) $\boldsymbol{\nu}q\,(p \leftrightarrow \bar{q} \mid \mathcal{A}[\![M]\!]_q) \gtrsim \mathcal{A}[\![M]\!]_p$
2) $\boldsymbol{\nu}x\,(x \leftrightarrow \bar{y} \mid \mathcal{A}[\![M]\!]_p) \gtrsim \mathcal{A}[\![M\{y/x\}]\!]_p$

B. Validity of β-reduction

The abstract encoding $\mathcal{A}$ validates β-reduction with respect to the expansion relation. This is proved by showing that substitution of a λ-term M is implemented as a communication to a replicated server that owns M. In the proofs of the following statements, Lemma IV.1 is frequently used.

Lemma IV.2. If $x \notin \mathsf{fv}(N)$, then
$$\boldsymbol{\nu}x\,(\mathcal{A}[\![M]\!]_p \mid !x(q).\,\mathcal{A}[\![N]\!]_q) \gtrsim \mathcal{A}[\![M\{N\!/x\}]\!]_p$$

Theorem IV.3. If $M \to N$, then $\mathcal{A}[\![M]\!]_p \gtrsim \mathcal{A}[\![N]\!]_p$.

Since bisimilarity is a congruence in Iπ and our encoding is compositional, the validity of β-reduction implies that the equivalence induced by the encoding is a λ-theory.

Corollary IV.4. Let $=_\pi \stackrel{\text{def}}{=} \{(M,N) \mid \mathcal{A}[\![M]\!] \approx \mathcal{A}[\![N]\!]\}$. Then $=_\pi$ is a λ-theory, that is, a congruence on λ-terms that contains β-equivalence.

Remark IV.1. From a λ-theory, a λ-model can be extracted [23], hence Corollary IV.4 implies that we can construct a λ-model out of the process terms. The domain of the model would be the processes that are in the image of the encoding, quotiented with bisimilarity. We could not define the domain of the model out of all process terms (as in [5], as opposed to the processes in the image of the encoding) because our proofs rely on Lemma IV.1, and such a lemma cannot be extended to the set of all processes.

V. Optimised Encoding

We introduce an optimised version of the abstract encoding, which removes certain 'administrative steps' on the process terms. This will allow us to have a sharper operational correspondence between λ-terms and processes, which will be needed in proofs in later sections. As in the previous section, we work with abstract wires, only assuming the requirements in Definition IV.1.

To motivate the need of the optimised encoding, let us consider the encoding of a term $(x\,M)\,N$:

$$(\boldsymbol{\nu}p_0, p_1)\,(\ \bar{x}(p_0')\colon p_0' \leftrightarrow \bar{p}_0$$
$$\mid \bar{p}_0(x_1.\,p_1')\colon (!x_1(r_1).\,[\![M]\!]_{r_1} \mid p_1 \leftrightarrow \bar{p}_1')$$
$$\mid \bar{p}_1(x_2, p_2)\colon (!x_2(r_2).\,[\![N]\!]_{r_2} \mid p \leftrightarrow \bar{p}_2)\ \ )$$

This process has, potentially (i.e., depending on the concrete instantiations of the wires), some initial administrative reductions. For instance, the output at p_1 may interact with the input end of the wire $p_1 \leftrightarrow \bar{p}_1'$.

In the optimised encoding $\mathcal{O}$, in Figure 2, any initial reduction of a process has a direct correspondence with a (strong call-by-name) reduction of the source λ-term. With respect to the unoptimised encoding $\mathcal{A}$, the novelties are in the clauses for application, where the case of a head normal form $x\,M_1 \cdots M_n$ (for $n \geq 1$) and of an application $(\lambda x.\,M_0)\,M_1 \cdots M_n$ with a head redex are distinguished. In both cases, $\mathcal{O}^n\langle p_0, p, \mathcal{O}[\![M_1]\!] \cdots \mathcal{O}[\![M_n]\!]\rangle$ is used for a compact representation of the encoding of the trailing arguments $M_1, \ldots, M_n$, as a sequence of nested permeable prefixes and a bunch of replications embracing the terms M_i.

Analogous properties to those in Section IV for the unoptimised encoding $\mathcal{A}$ hold for $\mathcal{O}$. Using such properties, and reasoning by induction of the structure of a λ-term, we can prove that $\mathcal{O}$ is indeed an optimisation.

Lemma V.1. $\mathcal{A}[\![M]\!]_p \gtrsim \mathcal{O}[\![M]\!]_p$.

The details about the operational behaviour of $\mathcal{O}[\![M]\!]_p$, and its operational correspondence with M, are described in [22]. We only report here the statements of a few important lemmas.

Lemma V.2. If $\mathcal{O}[\![M]\!]_p \xrightarrow{\tau} P$ then there exists N such that $M \to_{\mathtt{sn}} N$ and $P \gtrsim \mathcal{O}[\![N]\!]_p$.

Lemma V.3. If $\mathcal{O}[\![M]\!]_p \xrightarrow{\mu} P$ and μ is an input action, then μ is an input at p.

Later, when we relate our encoding to trees of the λ-calculus, the notions of head normal form and (un)solvable term will be important. Hence some of our operational correspondence results concern them.

Lemma V.4. Let M be a λ-term. If $\mathcal{O}[\![M]\!]_p \xrightarrow{\overline{x}(q)} P$ for some P, then M has a head normal form $\lambda\widetilde{y}.\,x\,\widetilde{M}$, for some (possibly empty) sequence of terms $\widetilde{M}$ and variables $\widetilde{y}$ with $x \notin \widetilde{y}$.

Thus, if $\mathcal{O}[\![M]\!]_p$ can perform an output action, then M is solvable.

Corollary V.5. M is solvable then there are input actions $\mu_1, \ldots, \mu_n$ $(n \geq 0)$ and an output action μ such that $\mathcal{O}[\![M]\!]_p \xRightarrow{\mu_1} \ldots \xRightarrow{\mu_n} \Rightarrow P$, for some P.

By Lemma V.1, Corollary V.5 also holds for $\mathcal{A}$. The converse of Corollary V.5 will also hold, in all three concrete encodings that will be studied in the next section. However we believe the result cannot be derived from the assumptions on wires in Definition IV.1.

We conclude by looking, as an example, at the unsolvable term $\Omega \stackrel{\text{def}}{=} (\lambda x.\,x\,x)\,(\lambda x.\,x\,x)$.

Example V.1. The process $\mathcal{O}[\![\Omega]\!]_p$ is

$$\boldsymbol{\nu}p_0(\ p_0(x, q)\colon \bar{x}(q_0)\colon \bar{q}_0(y_1, q_1)\colon (!y_1(r_1).\,\mathcal{O}[\![x]\!]_{r_1} \mid q \leftrightarrow \bar{q}_1)$$
$$\mid \bar{p}_0(x_1, p_1)\colon (!x_1(r_1).\,\mathcal{O}[\![\lambda x.\,x\,x]\!]_{r_1} \mid p \leftrightarrow \bar{p}_1))$$

The only action $\mathcal{O}[\![\Omega]\!]_p$ can do is a τ-action or an input at p. Whether the input can be performed or not will depend on the concrete definition of the wire $p \leftrightarrow \bar{q}$. The possibility of an input action from an unsolvable of order 0 such as Ω is a major difference between our encoding and encodings in the literature, where the encoding of such unsolvables are usually purely divergent processes.

VI. Concrete Wires

We now examine concrete instantiations of the abstract wires in the encoding $\mathcal{A}$ (and its optimisation $\mathcal{O}$). In each case we have to define the wires for location and variable names. The location wires are the important ones: the definition of the variable wires will follow from them, with the goal of guaranteeing their expected properties. We consider three concrete wires: *I-O wires*, *O-I wires*, and *P wires*. The main difference among them is in the order

$$\mathcal{O}[\![x]\!]_p \stackrel{\text{def}}{=} \bar{x}(p') \colon p \leftrightarrow \bar{p}'$$

$$\mathcal{O}[\![\lambda x.\, M]\!]_p \stackrel{\text{def}}{=} p(x,q) \colon \mathcal{O}[\![M]\!]_q$$

$$\mathcal{O}[\![x\, M_1 \cdots M_n]\!]_p \stackrel{\text{def}}{=} \bar{x}(p_0) \colon \mathcal{O}^n \langle p_0, p, \mathcal{O}[\![M_1]\!] \cdots \mathcal{O}[\![M_n]\!] \rangle$$

$$\mathcal{O}[\![(\lambda x.\, M_0)\, M_1 \cdots M_n]\!]_p \stackrel{\text{def}}{=} \boldsymbol{\nu} p_0\ (p_0(x,q) \colon \mathcal{O}[\![M_0]\!]_q \mid \mathcal{O}^n \langle p_0, p, \mathcal{O}[\![M_1]\!] \cdots \mathcal{O}[\![M_n]\!] \rangle)$$

$$\mathcal{O}^n \langle p_0, p, \mathcal{O}[\![M_1]\!] \cdots \mathcal{O}[\![M_n]\!] \rangle \stackrel{\text{def}}{=} \begin{aligned}[t] & \bar{p}_0(x_1, p_1) \colon \cdots \bar{p}_{n-1}(x_n, p_n) \colon \\ & (!x_1(r_1).\, \mathcal{O}[\![M_1]\!]_{r_1} \mid \cdots \mid !x_n(r_n).\, \mathcal{O}[\![M_n]\!]_{r_n} \mid p \leftrightarrow \bar{p}_n) \end{aligned}$$

Fig. 2. Optimised encoding. (The number n must be greater than 0 in the last three cases.)

in which the input and output of the location wires are performed.

Location and variable wires will be defined by means of mutual recursion. In contrast with the variable wires, the location wires are non-replicated processes, reflecting the linear usage of such names. We recall that the choice of a certain kind of concrete wires (I-O wires, O-I wires, or P wires) also affects the definition of the permeable prefixes (as it refers to the wires), including the permeable prefixes that may be used within the wires themselves. We will also show de-sugared definitions of the concrete wires, i.e., without reference to permeable prefixes. We add a subscript (IO, OI, P) to indicate a concrete wire (as opposed to an abstract one). For readability, in the definitions of the concrete wires the name parameters are instantiated (e.g., writing $a \underset{\text{IO}}{\leftrightarrow} \bar{b} \stackrel{\text{def}}{=} P$ rather than $\underset{\text{IO}}{\leftrightarrow} \stackrel{\text{def}}{=} (a,b)\, P$).

I-O wires: In the I-O wires, the input of a wire precedes the output.

$$p \underset{\text{IO}}{\leftrightarrow} \bar{q} \stackrel{\text{def}}{=} p(y, p_1).\, \bar{q}(x, q_1) \colon (p_1 \underset{\text{IO}}{\leftrightarrow} \bar{q}_1 \mid x \underset{\text{IO}}{\leftrightarrow} \bar{y})$$

$$x \underset{\text{IO}}{\leftrightarrow} \bar{y} \stackrel{\text{def}}{=} !x(p).\, \bar{y}(q) \colon p \underset{\text{IO}}{\leftrightarrow} \bar{q}$$

Inlining the abbreviations for permeable prefixes (as they are, in turn, defined using wires, in this specific case, the I-O wires), we obtain:

$$p \underset{\text{IO}}{\leftrightarrow} \bar{q} \stackrel{\text{def}}{=} \begin{aligned}[t] p(y, p_1).\, (\boldsymbol{\nu} x, q_1)\, (\bar{q}(x', q_1'). & (q_1 \underset{\text{IO}}{\leftrightarrow} \bar{q}_1' \mid x' \underset{\text{IO}}{\leftrightarrow} \bar{x}) \\ & \mid p_1 \underset{\text{IO}}{\leftrightarrow} \bar{q}_1 \mid x \underset{\text{IO}}{\leftrightarrow} \bar{y}) \end{aligned}$$

$$x \underset{\text{IO}}{\leftrightarrow} \bar{y} \stackrel{\text{def}}{=} !x(p).\, \boldsymbol{\nu} q\, (\bar{y}(q').\, q \underset{\text{IO}}{\leftrightarrow} \bar{q}' \mid p \underset{\text{IO}}{\leftrightarrow} \bar{q})$$

I-O wires, beginning with an input and proceeding with an output, are similar to the ordinary wires in the literature, sometimes called *forwarders*, and used to prove properties about asynchronous and localised π-calculi (or encodings of them) [13], [14], [16], [15]. An important technical difference, within location wires, is the appearance of a permeable prefix, in place of an ordinary prefix, and the inner wire $p_1 \underset{\text{IO}}{\leftrightarrow} \bar{q}_1$ that, in a forwarder, would have p_1 and q_1 swapped. The reason for these differences is that location wires are used with processes that are not

localised (the recipient of a location name will use it in input, rather than in output). The difference also shows up in the semantic properties: forwarders in the literature are normally used to obtain properties of O-respectfulness (Section III), with the input-end of the wire restricted; in contrast, I-O wires will be used to obtain properties of I-respectfulness, with the output-end of the wire restricted.

In the definition above of location wires, the permeable prefix cannot be replaced by an ordinary prefix: the transitivity of the wires (property 3 in Definitions IV.1) would be lost.

O-I wires: The symmetry of $I\pi$ enable us to consider the dual form of (location) wire, with the opposite control flow, namely 'from output to input':

$$p \underset{\text{OI}}{\leftrightarrow} \bar{q} \stackrel{\text{def}}{=} \bar{q}(x, q_1).\, p(y, p_1) \colon (p_1 \underset{\text{OI}}{\leftrightarrow} \overline{q_1} \mid x \underset{\text{OI}}{\leftrightarrow} \bar{y})$$

$$x \underset{\text{OI}}{\leftrightarrow} \bar{y} \stackrel{\text{def}}{=} !x(p).\, \bar{y}(q) \colon p \underset{\text{OI}}{\leftrightarrow} \bar{q}$$

Remark VI.1 (Duality). If duality is taken to mean the exchange between input and output prefixes, then the set of location I-O wires is the dual of the set of O-I wires. Indeed, the location O-I wires are obtained from the corresponding location I-O wires by swapping input and output particles; variable wires, in contrast are left unchanged. This means that we obtain an O-I wire from an I-O wire if any input $p(\widetilde{b})$ is made into an output $\bar{p}(\widetilde{b})$, and conversely (moreover, accordingly, the parameters of the variable wires are swapped).

P wires: In the third form of wire, the sequentiality in location wires is broken: input and output execute concurrently. This is achieved by using, in the definition of location wires, only permeable prefixes.

$$p \underset{\text{P}}{\leftrightarrow} \bar{q} \stackrel{\text{def}}{=} p(y, p_1) \colon \bar{q}(x, q_1) \colon (p_1 \underset{\text{P}}{\leftrightarrow} \bar{q}_1 \mid x \underset{\text{P}}{\leftrightarrow} \bar{y})$$

$$x \underset{\text{P}}{\leftrightarrow} \bar{y} \stackrel{\text{def}}{=} !x(p).\, \bar{y}(q) \colon p \underset{\text{P}}{\leftrightarrow} \bar{q}$$

Without the syntactic sugar of permeable prefixes, the definition of the location and variable P wires are thus:

$$p \underset{\text{P}}{\leftrightarrow} \bar{q} \stackrel{\text{def}}{=} \begin{aligned}[t] (\boldsymbol{\nu} p_1, q_1 x, y)\, (p(y', p_1'). & (p_1' \underset{\text{P}}{\leftrightarrow} \bar{p}_1 \mid y \underset{\text{P}}{\leftrightarrow} \bar{y}') \mid \\ & \bar{q}(q_1', x'). (q_1 \underset{\text{P}}{\leftrightarrow} \bar{q}_1' \mid x' \underset{\text{P}}{\leftrightarrow} \bar{x}) \mid \\ & p_1 \underset{\text{P}}{\leftrightarrow} \bar{q}_1 \mid x \underset{\text{P}}{\leftrightarrow} \bar{y}) \end{aligned}$$

$$x \underset{\mathsf{P}}{\leftrightarrow} \bar{y} \overset{\text{def}}{=} \; !x(p). \, \boldsymbol{\nu}q \, (\bar{y}(q'). \, q \underset{\mathsf{P}}{\leftrightarrow} \bar{q}' \mid p \underset{\mathsf{P}}{\leftrightarrow} \bar{q})$$

The wire $p \underset{\mathsf{P}}{\leftrightarrow} \bar{q}$ is dual of itself: due to the use of permeable prefixes, swapping input and output prefixes has no behavioural affect.

Lemma VI.1. The I-O wires, O-I wires and P wires satisfy the laws of Definition IV.1.

Proof. [Sketch] For each kind of wires, the proof is carried out in two steps. First, we show that the wires are transitive, using up-to techniques for bisimilarity. Then, the other laws are proved by algebraic reasoning (including the use of transitivity of wires). The proofs of transitivity are the most delicate ones, because of the concurrency allowed by permeable prefixes, especially in the case of P wires, and because permeable prefixes are defined in terms of the wires themselves. For example, unlike for forwarders in the literature, a wire $\boldsymbol{\nu}q(p \underset{\mathsf{P}}{\leftrightarrow} \bar{q} \mid q \underset{\mathsf{P}}{\leftrightarrow} \bar{r})$ can immediately reduce at the internal name q. Moreover, the derivative

$$p(p_1, y)\colon \bar{r}(q_1, x)\colon ((\boldsymbol{\nu}z_1, z_2) \, (x \underset{\mathsf{P}}{\leftrightarrow} \bar{z}_1 \mid z_1 \underset{\mathsf{P}}{\leftrightarrow} \bar{z}_2 \mid z_2 \underset{\mathsf{P}}{\leftrightarrow} \bar{y}) \mid$$
$$(\boldsymbol{\nu}s_1, s_2) \, (p_1 \underset{\mathsf{P}}{\leftrightarrow} \bar{s}_1 \mid s_1 \underset{\mathsf{P}}{\leftrightarrow} \bar{s}_2 \mid s_2 \underset{\mathsf{P}}{\leftrightarrow} \bar{q}_1))$$

shows that the reduction has made the chain of wires longer.

To prove transitivity, we crucially rely on up-to proof techniques for Iπ, notably 'expansion up-to expansion and context', and several algebraic laws. Thus we show that the relation with pairs of the form

$$\left(a_0 \underset{\mathsf{P}}{\leftrightarrow} \bar{a}_{n+1}, (\boldsymbol{\nu}a_1, \ldots, a_n) \, (a_0 \underset{\mathsf{P}}{\leftrightarrow} \bar{a}_1 \mid \cdots \mid a_n \underset{\mathsf{P}}{\leftrightarrow} \bar{a}_{n+1}) \right)$$

is an expansion up-to expansion and context. It is unclear how the proof could be carried out without such proof techniques. $\qquad\square$

Remark VI.2. The duality between I-O wires and O-I wires also shows up in proofs. For instance, for I-O wires the proof of law 4 of Definitions IV.1 does not use the premise of the law (i.e., the respectfulness of P), whereas the proof of the dual law 5 does. In the case of O-I wires, the opposite happens: the proof of law 5 uses the premise, whereas that of law 4 does not.

In the following sections we examine the concrete encodings obtained by instantiating the wires of the abstract encoding $\mathcal{A}$ (Figure 1) with the I-O wires, O-I wires, and P wires. We denote the resulting (concrete) encodings as $\mathcal{A}_{\mathtt{IO}}$, $\mathcal{A}_{\mathtt{OI}}$, and $\mathcal{A}_{\mathsf{P}}$, respectively. Similarly $\mathcal{O}_{\mathtt{IO}}$, $\mathcal{O}_{\mathtt{OI}}$, and $\mathcal{O}_{\mathsf{P}}$ are the instantiations of the abstract optimised encoding $\mathcal{O}$. For instance, in $\mathcal{A}_{\mathtt{IO}}$ and $\mathcal{O}_{\mathtt{IO}}$ an abstract wire $a \leftrightarrow \bar{b}$ is instantiated with the corresponding concrete wire $a \underset{\mathtt{IO}}{\leftrightarrow} \bar{b}$; and similarly for O-I wires and P wires.

Having shown that all the wires satisfy the requirements of Axiom IV.1, we can use, in the proofs about all concrete encodings (optimised and not) the results in Sections IV and V for the abstract encoding and its abstract optimisation.

VII. Full Abstraction for LTs and BTs

In this section we consider $\mathcal{A}_{\mathtt{IO}}$ and $\mathcal{A}_{\mathtt{OI}}$, and prove full abstraction with respect to the BTs and LTs, respectively. Before that, we discuss the difference between $\mathcal{A}_{\mathtt{IO}}$ and $\mathcal{A}_{\mathtt{OI}}$ on the encoding of unsolvable terms. In the proofs we exploit the optimised encodings $\mathcal{O}_{\mathtt{OI}}$ and $\mathcal{O}_{\mathtt{IO}}$. Details of the proofs are given in the full version [22].

We recall that the differences between BTs and LTs are due to the treatment of unsolvable terms (cf. Section II-A). BTs equate all the unsolvable terms, whereas LTs distinguish unsolvables of different order, such as Ω and $\lambda x. \Omega$. We begin, as an example, with the terms Ω and $\lambda x. \Omega$. As we have seen in Example V.1, in the abstract optimised encoding $\mathcal{O}$ process $\mathcal{O}[\![\Omega]\!]_p$ is:

$$\boldsymbol{\nu}p_0 \, (p_0(x, q)\colon \bar{x}(q_0)\colon \bar{q}_0(y_1, q_1)\colon (!y_1(r_1). \, \mathcal{O}[\![x]\!]_{r_1} \mid q \leftrightarrow \bar{q}_1)$$
$$\mid \bar{p}_0(x_1, p_1)\colon (!x_1(r_1). \, \mathcal{O}[\![\lambda x. \, x \, x]\!]_{r_1} \mid p \leftrightarrow \bar{p}_1)).$$

Its instantiation with O-I wires, $\mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_p$, cannot do any input action: as $p \leftrightarrow \bar{p}_1$ becomes the O-I wire $p \underset{\mathtt{OI}}{\leftrightarrow} \overline{p_1}$, the input occurrence of the free name p is guarded by p_1, which in turn is bound by the (permeable) prefix at p_0. Indeed, the only action that $\mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_p$ can perform is (up-to expansion) $\mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_p \overset{\tau}{\to} \gtrsim \mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_p$, which corresponds to the reduction $\Omega \to \Omega$. Hence, $\mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_p$ cannot match the input action $\mathcal{O}_{\mathtt{OI}}[\![\lambda x. \, \Omega]\!]_p \xrightarrow{p(x,q)} \mathcal{O}_{\mathtt{OI}}[\![\Omega]\!]_q$, and the two processes are distinguished.

In contrast, with I-O wires, processes $\mathcal{O}_{\mathtt{IO}}[\![\lambda x. \, \Omega]\!]_p$ and $\mathcal{O}_{\mathtt{IO}}[\![\Omega]\!]_p$ are indistinguishable. As before, the former process can exhibit an input transition $\mathcal{O}_{\mathtt{IO}}[\![\lambda x. \, \Omega]\!]_p \xrightarrow{p(x,q)} \mathcal{O}_{\mathtt{IO}}[\![\Omega]\!]_q$. However now $\mathcal{O}_{\mathtt{IO}}[\![\Omega]\!]_p$ has a matching input transition, because when $p \leftrightarrow \bar{p}_1$ is the I-O wire $p \underset{\mathtt{IO}}{\leftrightarrow} \bar{p}_1$, the input at p is not guarded. The derivative is

$$\boldsymbol{\nu}p_0 \, (p_0(x, q)\colon \mathcal{O}_{\mathtt{IO}}[\![x \, x]\!]_q$$
$$\mid \bar{p}_0(x_1, p_1)\colon (!x_1(r_1). \, \mathcal{O}_{\mathtt{IO}}[\![\lambda x. \, x \, x]\!]_{r_1}$$
$$\mid \bar{p}_1(x_2, p_2)\colon (x_2 \underset{\mathtt{IO}}{\leftrightarrow} \bar{y} \mid q \underset{\mathtt{IO}}{\leftrightarrow} \bar{p}_2)))$$
$$= \boldsymbol{\nu}p_0 \, (p_0(x, q)\colon \mathcal{O}_{\mathtt{IO}}[\![x \, x]\!]_q$$
$$\mid \bar{p}_0(x_1, p_1)\colon (!x_1(r_1). \, \mathcal{O}_{\mathtt{IO}}[\![\lambda x. \, x \, x]\!]_{r_1}$$
$$\mid \bar{p}_1(x_2, p_2)\colon (!x_2(r_2). \, \mathcal{O}_{\mathtt{IO}}[\![y]\!]_{r_2} \mid q \underset{\mathtt{IO}}{\leftrightarrow} \bar{p}_2)))$$
$$= \mathcal{O}_{\mathtt{IO}}[\![\Omega \, y]\!]_q$$

(exploiting the definitions of $\mathcal{O}_{\mathtt{IO}}[\![y]\!]_{r_2}$ and $x_2 \underset{\mathtt{IO}}{\leftrightarrow} \bar{y}$). In a similar manner, one then shows that $\mathcal{O}_{\mathtt{IO}}[\![\Omega]\!]_q$ and $\mathcal{O}_{\mathtt{IO}}[\![\Omega \, y]\!]_q$ can match each other's transitions, and iteratively so, on the resulting derivatives.

More generally, only in $\mathcal{O}_{\mathtt{OI}}$ a term $\mathcal{O}_{\mathtt{OI}}[\![M]\!]_p$ can perform an input transition if and only if M is, or may reduce to, a function, say $M = \lambda x. \, M'$, and the input action intuitively corresponds to consuming the outermost 'λx'. In addition, only with $\mathcal{O}_{\mathtt{OI}}$ a process $\mathcal{O}_{\mathtt{OI}}[\![M]\!]_p$ is bisimilar to $\mathbf{0}$ iff the term M is an unsolvable of order 0. Therefore, we have:

Lemma VII.1. Let M and N be unsolvables of order m and n respectively, where $0 \leq m, n \leq \omega$. Then $\mathcal{O}_{\mathtt{OI}}[\![M]\!]_p \approx \mathcal{O}_{\mathtt{OI}}[\![N]\!]_p$ iff $m = n$.

We have discussed above why, in contrast, $\mathcal{O}_{\mathtt{IO}}$ equates $\lambda x. \Omega$ and Ω. Similarly, $\mathcal{O}_{\mathtt{IO}}$ equates all the unsolvable terms.

Lemma VII.2. For any unsolvable term M, we have $\mathcal{O}_{\mathtt{IO}}[\![M]\!]_p \approx \mathcal{O}_{\mathtt{IO}}[\![\Omega]\!]_p$.

Proof. [Sketch] We show that the relation defined as $\{(\mathcal{O}_{\mathtt{IO}}[\![M]\!]_p, \mathcal{O}_{\mathtt{IO}}[\![N]\!]_p) \mid M, N \text{ are unsolvable}\}$ is a bisimulation up-to expansion. For this, we use Lemmas V.2, and V.3, and exploit the property that, for any unsolvable M, it holds that $\mathcal{O}_{\mathtt{IO}}[\![M]\!]_p \xrightarrow{p(x,q)} \approx \mathcal{O}_{\mathtt{IO}}[\![M\,x]\!]_q$, where $M\,x$ is unsolvable since M is so. $\square$

Theorem VII.3. [Full abstraction for LT and BT] For every λ-terms M and N, we have:

1) $\mathrm{LT}(M) = \mathrm{LT}(N)$ if and only if $\mathcal{A}_{\mathtt{OI}}[\![M]\!] \approx \mathcal{A}_{\mathtt{OI}}[\![N]\!]$.
2) $\mathrm{BT}(M) = \mathrm{BT}(N)$ if and only if $\mathcal{A}_{\mathtt{IO}}[\![M]\!] \approx \mathcal{A}_{\mathtt{IO}}[\![N]\!]$.

Proof. [Sketch] The proofs exploit [6], which sets conditions for obtaining full abstraction with respect to LTs and BTs in an encoding of the λ-calculus into a process calculus. The conditions refer to the behavioural equivalence of the process calculus and to an auxiliary behavioural preorder contained in the equivalence. Our proofs go through each such condition, showing that it is satisfied. Below we report some of the conditions, tailored to our setting, where the encodings are $\mathcal{A}_{\mathtt{OI}}$ and $\mathcal{A}_{\mathtt{IO}}$, the calculus is I$\pi$, behavioural equivalence is $\approx$, and the auxiliary preorder is $\lesssim$. Then, we briefly comment on their proof. We refer to [22] for more details. Some conditions are common to LTs and BTs, and include:

(i) the encoding validates the β rule with respect to $\lesssim$;
(ii) the encodings of the terms terms Ω, $x\,\widetilde{M}$, $x\,\widetilde{M'}$, and $y\,\widetilde{M''}$ are pairwise unrelated by $\approx$, assuming that $x \neq y$ and that tuples $\widetilde{M}$ and $\widetilde{M'}$ have different lengths.

The conditions specific to LTs are:

LT-i) $\mathcal{A}_{\mathtt{OI}}[\![M]\!] \approx \mathcal{A}_{\mathtt{OI}}[\![N]\!]$, for all unsolvables M, N of order ω;
LT-ii) for any M, the term $\mathcal{A}_{\mathtt{OI}}[\![\lambda x. M]\!]$ is unrelated by $\approx$ to $\mathcal{A}_{\mathtt{OI}}[\![\Omega]\!]$ and to any term of the form $\mathcal{A}_{\mathtt{OI}}[\![x\,\widetilde{M}]\!]$.

The corresponding conditions for BTs are:

BT-i) $\mathcal{A}_{\mathtt{IO}}[\![M]\!] \approx \mathcal{A}_{\mathtt{IO}}[\![\Omega]\!]$ for all unsolvable M of order ω;
BT-ii) M solvable implies that the term $\mathcal{A}_{\mathtt{IO}}[\![\lambda x. M]\!]$ is unrelated by $\approx$ to $\mathcal{A}_{\mathtt{IO}}[\![\Omega]\!]$ and to any term of the form $\mathcal{A}_{\mathtt{IO}}[\![x\,\widetilde{M}]\!]$.

Some of the common conditions can be proved at the abstract level, for $\mathcal{A}$ or its optimisation $\mathcal{O}$, and are therefore valid for both $\mathcal{A}_{\mathtt{OI}}$ and $\mathcal{A}_{\mathtt{IO}}$: for instance, validity of β-reduction for $\lesssim$ is Theorem IV.3; and the behavioural

difference among (the encodings of) Ω, $x\,\widetilde{M}$, and $y\,\widetilde{M''}$ is obtained from the operational correspondence results for the optimised abstract encoding $\mathcal{O}$.

The conditions (LT-i) and (BT-i) are about unsolvable terms. We have seen (Lemmas VII.1 and VII.2) that $\mathcal{O}_{\mathtt{OI}}$ (hence also $\mathcal{A}_{\mathtt{OI}}$) satisfies (LT-i) and that $\mathcal{O}_{\mathtt{IO}}$ (hence also $\mathcal{A}_{\mathtt{IO}}$) satisfies (BT-i). A delicate condition to check is (BT-ii) for $\mathcal{A}_{\mathtt{IO}}$. For this, once more we exploit the optimised encoding $\mathcal{O}_{\mathtt{IO}}$ and reason on the number of consecutive outputs that the processes can perform. $\square$

Remark VII.1. Neither $\mathcal{A}_{\mathtt{IO}}$ nor $\mathcal{A}_{\mathtt{OI}}$ validates the η-rule (i.e., the λ-theories induced are not extensional); this follows from Theorem VII.3 (specifically, conditions (LT-ii) and (BT-ii) mentioned in its proof).

VIII. Full Abstraction for $\mathrm{BT}_{\eta\infty}$s

In this section we show that the encoding $\mathcal{A}_{\mathtt{P}}$ obtained by instantiating the wires of the abstract encoding $\mathcal{A}$ of Section IV with the parallel wires (the P wires) yields an encoding that is fully abstract with respect to $\mathrm{BT}_{\eta\infty}$s (Böhm trees with infinite η-expansion).

We begin by showing that $\mathcal{A}_{\mathtt{P}}$ induces an *extensional* λ-theory. As we know (Section VI) that $\mathcal{A}_{\mathtt{P}}$ induces a λ-theory, we remain to check the validity of η-expansion.

Theorem VIII.1. For every M and $x \notin \mathsf{fv}(M)$, we have $\mathcal{A}_{\mathtt{P}}[\![M]\!]_p \lesssim \mathcal{A}_{\mathtt{P}}[\![\lambda x. M\,x]\!]_p$.

Proof. The process $\mathcal{A}_{\mathtt{P}}[\![\lambda x. M\,x]\!]_p$ is

$$p(x,q)\colon \boldsymbol{\nu}r\,(\mathcal{A}_{\mathtt{P}}[\![M]\!]_r \mid \bar{r}(x',q')\colon (!x'(r').\mathcal{A}_{\mathtt{P}}[\![x]\!]_{r'} \mid q \underset{\mathtt{P}}{\leftrightarrow} \bar{q}')).$$

As $!x'(r').\mathcal{A}_{\mathtt{P}}[\![x]\!]_{r'} = x' \underset{\mathtt{P}}{\leftrightarrow} \bar{x}$, we have:

$$\begin{aligned}
&\mathcal{A}_{\mathtt{P}}[\![\lambda x. M\,x]\!]_p \\
&= \boldsymbol{\nu}r\,(\mathcal{A}_{\mathtt{P}}[\![M]\!]_r \mid p(x,q)\colon \bar{r}(x',q')\colon (x' \underset{\mathtt{P}}{\leftrightarrow} \bar{x} \mid q \underset{\mathtt{P}}{\leftrightarrow} \bar{q}')) \\
&= \boldsymbol{\nu}r\,(\mathcal{A}_{\mathtt{P}}[\![M]\!]_r \mid p \underset{\mathtt{P}}{\leftrightarrow} \bar{r}) \gtrsim \mathcal{A}_{\mathtt{P}}[\![M]\!]_p
\end{aligned}$$

using Lemma IV.1 (and identifying processes 'up-to structure'). $\square$

The above result relies on the use of permeable prefixes, both in the encoding of λ-abstraction, and within the P wires.

Corollary VIII.2. Let $=_\pi \overset{\mathrm{def}}{=} \{(M,N) \mid \mathcal{A}_{\mathtt{P}}[\![M]\!] \approx \mathcal{A}_{\mathtt{P}}[\![N]\!]\}$. Then $=_\pi$ is an extensional λ-theory; that is, a congruence on λ-terms that contains β and η-equivalence.

We are now ready to prove that $\mathcal{A}_{\mathtt{P}}$ is fully abstract with respect to $\mathrm{BT}_{\eta\infty}$s. We focus on completeness: if $\mathrm{BT}_{\eta\infty}(M) = \mathrm{BT}_{\eta\infty}(N)$ then $\mathcal{A}_{\mathtt{P}}[\![M]\!] \approx \mathcal{A}_{\mathtt{P}}[\![N]\!]$. Soundness will then be essentially derived from completeness, as $\mathrm{BT}_{\eta\infty}$ equality is the maximal consistent sensible λ-theory (see e.g. [23]). To show completeness, we rely on the 'unique solution of equation' technique, reviewed in Section II-B1.

Remark VIII.1 (Unique solutions versus up-to techniques). Results about encodings of λ-calculus into process calculi, in previous sections of this paper and in the literature, usually employ up-to techniques for bisimilarity, notably *up-to context and expansion*. In the techniques, expansion is used to manipulate the derivatives of two transitions so to bring up a common context. Such techniques do not seem powerful enough for $\mathrm{BT}_{\eta\infty}$. The reason is that some of the required transformations would violate expansion (i.e., they would require to replace a term by a 'less efficient' one), for instance 'η-expanding' a term $\mathcal{A}_{\mathsf{P}}[\![z]\!]_p$ into $\mathcal{A}_{\mathsf{P}}[\![\lambda\, y.\, z\, y]\!]_p$. A similar problem has been observed in the case of Milner's call-by-value encoding [27].

Suppose $\mathcal{R}$ is a $\mathrm{BT}_{\eta\infty}$-bisimulation (Definition II.2). We define a (possibly infinite) system of equations $\mathcal{E}_{\mathcal{R}}$, solutions of which will be obtained from the encodings of the pairs in $\mathcal{R}$. There is one equation for each pair $(M, N) \in \mathcal{R}$. We describe how each equation is defined, following the clauses of $\mathrm{BT}_{\eta\infty}$-bisimulation. Take $(M, N) \in \mathcal{R}$ and assume $\widetilde{y} = \mathsf{fv}(M, N)$.

1) If M and N are unsolvable, then, for the right-hand side of the equation, we pick a non-divergent process that is bisimilar to the encoding of Ω:

$$X_{M,N}\, \widetilde{y} = \mathsf{K}_{\Omega}$$

For instance, we may choose $\mathsf{K}_{\Omega} \stackrel{\text{def}}{=} (p)\, p(x, q)\colon \mathsf{K}_{\Omega}\langle q\rangle$.

2) If $M \Longrightarrow_{\mathrm{h}} \lambda x_1 \ldots x_{l+m}.\, z\, M_1 \cdots M_{n+m}$ and $N \Longrightarrow_{\mathrm{h}} \lambda x_1 \ldots x_l.\, z\, N_1 \cdots N_n$, then the equation is:

$$X_{M,N}\, \widetilde{y}\, p \stackrel{\text{def}}{=}$$
$$p(x_1, p_1)\colon \cdots p_{l+m-1}(x_{l+m}, p_{l+m})\colon \bar{z}(w, q)\colon$$
$$\mathcal{O}_{\mathsf{P}}^{n+m} \left\langle \begin{matrix} q, p_{l+m}, X_{M_1, N_1}\langle\widetilde{y_1}\rangle, \ldots, X_{M_n, N_n}\langle\widetilde{y_n}\rangle, \\ X_{M_{n+1}, x_{l+1}}\langle\widetilde{y_{n+1}}\rangle, \ldots, X_{M_{n+m}, x_{l+m}}\langle\widetilde{y_{n+m}}\rangle \end{matrix} \right\rangle$$

where $\quad \widetilde{y_i} = \mathsf{fv}(M_i, N_i)$ for $1 \leq i \leq n$,
$\quad\quad \widetilde{y_i} = \mathsf{fv}(M_i, x_{i-n+l})$ for $n+1 \leq i \leq n+m$.
and where $\mathcal{O}_{\mathsf{P}}^r$ is the instantiation with P wires of $\mathcal{O}^r$ in Figure 2.

3) For the case symmetric to (2), where N reduces to a head normal form with more leading λ-abstractions, the equation is defined similarly to (2).

In (1), the use of a divergent-free term K_{Ω} allows us to meet the condition about divergence of the unique-solution technique. The right-hand side of (2) intuitively amounts to having, as a body of the equation, the process $\mathcal{O}_{\mathsf{P}}[\![\lambda x_1 \ldots x_{l+m}.\, z\, X_{M_1, N_1} \cdots X_{M_{n+m}, x_{l+m}}]\!]$.

Lemma VIII.3. For any M unsolvable, we have:
$\mathcal{O}_{\mathsf{P}}[\![M]\!]_p \approx \mathcal{O}_{\mathsf{P}}[\![\Omega]\!]_p \approx \mathsf{K}_{\Omega}\langle p\rangle$.

Lemma VIII.4. Let $\mathcal{R}$ be a $\mathrm{BT}_{\eta\infty}$-bisimulation and $\mathcal{E}_{\mathcal{R}}$ be the system of equations defined from $\mathcal{R}$ as above. For each $(M, N) \in \mathcal{R}$, we define $F_{M,N} \stackrel{\text{def}}{=} (\widetilde{x}, p)\, \mathcal{O}_{\mathsf{P}}[\![M]\!]_p$ and $G_{M,N} \stackrel{\text{def}}{=} (\widetilde{x}, p)\, \mathcal{O}_{\mathsf{P}}[\![N]\!]_p$, where $\widetilde{x} = \mathsf{fv}(M, N)$. Then $\{F_{M,N}\}_{(M,N)\in\mathcal{R}}$ and $\{G_{M,N}\}_{(M,N)\in\mathcal{R}}$ are solutions of $\mathcal{E}_{\mathcal{R}}$.

Proof. [Sketch] There are three cases to consider, following Definition II.2. The case of M and N unsolvables is handled via Lemma VIII.3. The case of solvable term is reported in [22] and exploits validity of the η-expansion (Theorem VIII.1), the results of operational correspondence for the optimised encoding discussed in Section V, and other algebraic reasoning. $\qquad\square$

We also have to show that the system $\mathcal{E}_{\mathcal{R}}$ of equations we defined has a unique solution.

Lemma VIII.5. The system of equations $\mathcal{E}_{\mathcal{R}}$ is guarded and the syntactic solution of $\mathcal{E}_{\mathcal{R}}$ is divergence-free. Therefore, $\mathcal{E}_{\mathcal{R}}$ has a unique solution.

Proof. The system $\mathcal{E}_{\mathcal{R}}$ is guarded because all the occurrences of a variable in the right-hand side of an equation are underneath a replicated input prefixing. Divergence-freedom follows from the fact that the use of each name (bound or free) is strictly polarised in the sense that a name is either used as an input or as an output. In a strictly polarised setting, no τ-transitions can be performed even after some visible actions because in $\mathrm{I}\pi$ only fresh names may be exchanged. $\qquad\square$

Theorem VIII.6 (Completeness for $\mathrm{BT}_{\eta\infty}$). If $\mathrm{BT}_{\eta\infty}(M) = \mathrm{BT}_{\eta\infty}(N)$ then $\mathcal{A}_{\mathsf{P}}[\![M]\!] \approx \mathcal{A}_{\mathsf{P}}[\![N]\!]$.

Proof. Consider a $\mathrm{BT}_{\eta\infty}$-bisimulation $\mathcal{R}$ that equates M and N. Take the system of equations $\mathcal{E}_{\mathcal{R}}$ corresponding to $\mathcal{R}$ as defined above. By Lemma VIII.4, $\mathcal{O}_{\mathsf{P}}[\![M]\!]$ and $\mathcal{O}_{\mathsf{P}}[\![N]\!]$ are (components) of the solutions of $\mathcal{E}_{\mathcal{R}}$. Since the solution is unique (Lemma VIII.5), we derive $\mathcal{O}_{\mathsf{P}}[\![M]\!] \approx \mathcal{O}_{\mathsf{P}}[\![N]\!]$. We also have $\mathcal{A}_{\mathsf{P}}[\![M]\!] \approx \mathcal{A}_{\mathsf{P}}[\![N]\!]$ (equivalence on the non-optimised encodings) because of Lemma V.1. $\qquad\square$

Theorem VIII.7 (Soundness for $\mathrm{BT}_{\eta\infty}$). If $\mathcal{A}_{\mathsf{P}}[\![M]\!] \approx \mathcal{A}_{\mathsf{P}}[\![N]\!]$ then $\mathrm{BT}_{\eta\infty}(M) = \mathrm{BT}_{\eta\infty}(N)$.

Proof. Let $=_{\pi}$ be the equivalence induced by $\mathcal{A}_{\mathsf{P}}$ and $\mathrm{I}\pi$ bisimilarity. The equivalence $=_{\pi}$ is a *sensible* λ-theory by Corollary IV.4 and Lemma VIII.3. This theory is consistent: for example we have $\mathcal{A}_{\mathsf{P}}[\![x]\!]_p \not\approx \mathcal{A}_{\mathsf{P}}[\![\Omega]\!]_p$. By completeness (Theorem VIII.6), it contains $\mathrm{BT}_{\eta\infty}$ equality. Then it must be equal to $\mathrm{BT}_{\eta\infty}$ equality because the latter is the maximal consistent sensible λ-theory [23]. $\qquad\square$

IX. Concluding Remarks

In the paper we have presented a refinement of Milner's original encoding of functions as processes that is parametric on certain abstract components called wires. Whenever wires satisfy a few algebraic properties, the encoding yields a λ-theory. We have studied instantiations of the abstract wires with three kinds of concrete wires, that differ on the direction and/or sequentiality of the control flow produced. We have shown that such instantiations allow us to obtain full abstraction results for LTs, BTs, and $\mathrm{BT}_{\eta\infty}$s, (and hence for λ-models such as P_{ω}, free lazy Plotkin-Scott-Engeler models and D_{∞}). In the case of

$BT_{\eta\infty}$, this implies that the encoding validates the η-rule, i.e., it yields an extensional λ-theory.

Following Milner's seminal paper [1], the topic of functions as processes has produced a rich bibliography. Below we comment on the works that seem closest to ours. We have mentioned, in the introduction, related work concerning LTs and BTs. We are unaware of results about validity of the η-rule, let alone $BT_{\eta\infty}$, in encodings of functions as processes. The only exception is [28], where a type system for the π-calculus is introduced so to derive full abstraction for an encoding of PCF (which implies that η-expansion for PCF is valid). However, in [28], types are used to constrain process behaviours, so to remain with processes that represent 'sequential functional computations'. Accordingly, the behavioural equivalence for processes is a typed contextual equivalence in which the legal contexts must respect the typing discipline and are therefore 'sequential'. In contrast, in our work η is validated under ordinary (unconstrained) process equivalence in which, for instance, equalities are preserved by arbitrary process contexts. (We still admit polyadic communications and hence a sorting system, for readability — we believe that the same results hold in a monadic setting.)

In the paper we have considered the theory of the pure untyped λ-calculus. Hence, our encodings model the call-by-name reduction strategy. A study of the theory induced by process encodings of the call-by-value strategy is [27].

Our definitions and proofs about encodings of permeable prefixes using wires follows, and is inspired by, encodings of forms of permeable prefixes in asynchronous and localised variants of the π-calculus using forwarders, e.g. [16], [29]. As commented in the main text, the technicalities are however different, both because our processes not localised, and because we employ distinct kinds of wires.

We have worked with bisimilarity, as it is the standard behavioural equivalence in Iπ; moreover, we could then use some powerful proof techniques for it (up-to techniques, unique solution of equations). The results presented also hold for other behavioural equivalences (e.g., may testing), since processes encoding functions are confluent. It would be interesting to extend our work to preorders, i.e., looking at preorder relations for λ-trees and λ-models.

In our work, we derived our abstract encoding from Milner's original encoding of functions. It is unclear how to transport the same methodology to other variants of Milner's encoding in the literature, in particular those that closely mimics the CPS translations [30], [31].

We have derived λ-tree equalities, in parametric manner, by different instantiations of the abstract wires. Van Bakel et al. [32] use an intersection type system, parametric with respect to the subtyping relations, to (almost) uniformly characterise λ-tree equalities (the trees considered are those in our paper together with Böhm trees up-to *finite* η-expansion and Beraducci trees).

We would like to investigate the possible relationship between our work and game semantics. In particular, we are interested in the 'HO/N style' as it is known to be related to process representations (e.g., [33], [34], [35], [36]). HO/N game semantics for the three trees considered in this paper have been proposed [37], [38], [39]. The technical differences with our work are substantial. For instance, the game semantics are not given in a parametric manner; and the D_∞ equality is obtained via Nakajima trees rather than $BT_{\eta\infty}$. Nakajima trees are a different 'infinite η-expansion' of Böhm trees, in the sense that $\lambda\widetilde{x}.\,y\,\widetilde{M}$ is expanded to $\lambda\widetilde{x}z_0z_1\ldots.\,y\,\widetilde{M}\,z_0z_1\cdots$; that is, trees may be infinitely branching. In processes, this would mean, for instance, having input prefixes that receive infinitely-many names at the same time. We would like to understand whether the three kinds of wires we considered are meaningful in game semantics. The game semantic counterpart of process wires are the *copycat strategies*, and they intuitively correspond to I-O wires, in that they begin with an O-move (i.e., an input action). This does not change even in concurrent game semantics [40], [41]. We are not aware of game models that use strategies corresponding to the O-I wires or the P wires studied in our paper.

Similarly, we would like to investigate relationships with call-by-name translations of the λ-calculus into (pure) proof-nets [42]. We think that our encoding could be factorised into the translation from λ-calculus into proof-nets and a variant of Abramsky translation [18], [19]. In this way, a P wire for location names would correspond to an infinitely η-expanded form of the axiom link for the type o according to its recursive equation $o \cong (!o)^\perp \,\mathbin{⅋}\, o$. Infinite η-expansions of the identity axioms have also been considered in Girard's ludics [43], where they are called faxes. Faxes are different from P wires because faxes satisfy an alternation condition akin to the locality of π-calculi.

Processes like wires (often called *links*) appear in session-typed process calculi focusing on the Curry-Howard isomorphism, as primitive process constructs used to represent the identity axiom [44]. Some of our assumptions for wires, cf. the substitution-like behaviour when an end-point of the wire is restricted, are then given as explicit rules of the operational semantics of such links.

We have studied the properties of the concrete wires used in the paper on processes encoding functions. We would like to establish more general properties, on arbitrary processes, possibly subject to constraints on the usage of the names of the wires. We would also like to see if other kinds of wires are possible, and which properties they yield.

References

[1] R. Milner, "Functions as processes," INRIA, Research Report RR-1154, 1990. [Online]. Available: https://hal.inria.fr/inria-00075405

[2] ——, "Functions as processes," *Math. Struct. Comput. Sci.*, vol. 2, no. 2, pp. 119–141, 1992.

[3] D. Sangiorgi and D. Walker, *The π-calculus—A Theory of Mobile Processes.* Cambridge University Press, 2001.

[4] D. Sangiorgi, "An investigation into functions as processes," in *Mathematical Foundations of Programming Semantics (MFPS)*, ser. Lecture Notes in Computer Science, S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, Eds., vol. 802. Springer, 1993, pp. 143–159.

[5] ——, "Lazy functions and mobile processes," in *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, G. D. Plotkin, C. Stirling, and M. Tofte, Eds. The MIT Press, 2000, pp. 691–720.

[6] D. Sangiorgi and X. Xu, "Trees from functions as processes," *Log. Methods Comput. Sci.*, vol. 14, no. 3, 2018.

[7] G. Plotkin, "A set theoretical definition of application," School of A.I., Univ. of Edinburgh, Tech. Rep. Tech. Rep. MIP-R-95, 1972.

[8] D. S. Scott, "Data types as lattices," *SIAM J. Comput.*, vol. 5, no. 3, pp. 522–587, 1976.

[9] J. Lévy, "An algebraic interpretation of the *lambda beta* k-calculus; and an application of a labelled *lambda* -calculus," *Theor. Comput. Sci.*, vol. 2, no. 1, pp. 97–114, 1976.

[10] E. Engeler, "Algebras and combinators," *Algebra Universalis*, vol. 13, pp. 389–392, 1981.

[11] G. Longo, "Set theoretical models of lambda calculus: Theory, expansions and isomorphisms," *Annales of Pure and Applied Logic*, vol. 24, pp. 153–188, 1983.

[12] C. P. Wadsworth, "The relation between computational and denotational properties for scott's d_{infty}-models of the lambda-calculus," *SIAM J. Comput.*, vol. 5, no. 3, pp. 488–521, 1976.

[13] K. Honda and N. Yoshida, "On reduction-based process semantics," *Theor. Comput. Sci.*, vol. 152, no. 2, pp. 437–486, 1995.

[14] M. Merro, "Locality in the π-calculus and applications to object-oriented languages," 2001, PhD thesis, Ecoles des Mines de Paris.

[15] M. Boreale, "On the expressiveness of internal mobility in name-passing calculi," *Theor. Comput. Sci.*, vol. 195, no. 2, pp. 205–226, 1998.

[16] M. Merro and D. Sangiorgi, "On asynchrony in name-passing calculi," *Mathematical Structures in Computer Science*, vol. 14, no. 5, pp. 715–767, 2004, a preliminary version in Proc. ICALP'98.

[17] L. Caires, F. Pfenning, and B. Toninho, "Linear logic propositions as session types," *Mathematical Structures in Computer Science*, vol. 26, no. 3, pp. 367–423, 2016.

[18] S. Abramsky, "Proofs as processes," *Theor. Comput. Sci.*, vol. 135, no. 1, pp. 5–9, 1994.

[19] G. Bellin and P. J. Scott, "On the pi-calculus and linear logic," *Theor. Comput. Sci.*, vol. 135, no. 1, pp. 11–65, 1994.

[20] Y. Fu, "A proof theoretical approach to communication," in *24th ICALP*, ser. Lecture Notes in Computer Science, vol. 1256. Springer Verlag, 1997.

[21] J. Parrow and B. Victor, "The fusion calculus: Expressiveness and symmetry in mobile processes," in *13th LICS Conf.* IEEE Computer Society Press, 1998.

[22] K. Sakayori and D. Sangiorgi, "Extensional and non-extensional functions as processes (long version)." [Online]. Available: https://hal.science/hal-04081885

[23] H. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, ser. North-Holland Linguistic Series. North-Holland, 1984.

[24] S. B. Lassen, "Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context," in *Mathematical Foundations of Progamming Semantics MFPS 1999*, ser. Electronic Notes in Theoretical Computer Science, S. D. Brookes, A. Jung, M. W. Mislove, and A. Scedrov, Eds., vol. 20. Elsevier, 1999, pp. 346–374.

[25] R. Milner, "The polyadic π-calculus: a tutorial," in *Logic and Algebra of Specification*, ser. NATO ASI Series, F. Bauer, W. Brauer, and H. Schwichtenberg, Eds. Springer Berlin Heidelberg, 1993, vol. 94, pp. 203–246.

[26] A. Durier, D. Hirschkoff, and D. Sangiorgi, "Divergence and unique solution of equations," *Log. Methods Comput. Sci.*, vol. 15, no. 3, 2019.

[27] ——, "Eager functions as processes," *Theor. Comput. Sci.*, vol. 913, pp. 8–42, 2022.

[28] M. Berger, K. Honda, and N. Yoshida, "Sequentiality and the pi-calculus," in *Typed Lambda Calculi and Applications TLCA 2001*, ser. Lecture Notes in Computer Science, S. Abramsky, Ed., vol. 2044. Springer, 2001, pp. 29–45.

[29] N. Yoshida, "Minimality and separation results on asynchronous mobile processes - representability theorems by concurrent combinators," *Theor. Comput. Sci.*, vol. 274, no. 1-2, pp. 231–276, 2002.

[30] D. Sangiorgi, "From λ to π; or, Rediscovering continuations," *Mathematical Structures in Computer Science*, vol. 9, no. 4, pp. 367–401, 1999.

[31] H. Thielecke, "Categorical structure of continuation passing style," Ph.D. dissertation, University of Edinburgh, UK, 1997.

[32] S. van Bakel, F. Barbanera, M. Dezani-Ciancaglini, and F. de Vries, "Intersection types for lambda-trees," *Theor. Comput. Sci.*, vol. 272, no. 1-2, pp. 3–40, 2002.

[33] J. M. E. Hyland and C. L. Ong, "Pi-calculus, dialogue games and PCF," in *Proceedings of conf. on Functional programming languages and computer architecture (FPCA)*, J. Williams, Ed. ACM, 1995, pp. 96–107.

[34] K. Honda and N. Yoshida, "Game-theoretic analysis of call-by-value computation," *Theor. Comput. Sci.*, vol. 221, no. 1-2, pp. 393–456, 1999.

[35] S. Castellan and N. Yoshida, "Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 27:1–27:29, 2019.

[36] G. Jaber and D. Sangiorgi, "Games, mobile processes, and functions," in *30th EACSL Annual Conference on Computer Science Logic, CSL 2022*, ser. LIPIcs, F. Manea and A. Simpson, Eds., vol. 216. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 25:1–25:18.

[37] A. D. Ker, H. Nickau, and C. L. Ong, "Innocent game models of untyped lambda-calculus," *Theor. Comput. Sci.*, vol. 272, no. 1-2, pp. 247–292, 2002.

[38] ——, "Adapting innocent game models for the Böhm tree lambda-theory," *Theor. Comput. Sci.*, vol. 308, no. 1-3, pp. 333–366, 2003.

[39] C. L. Ong and P. D. Gianantonio, "Games characterizing Levy-Longo trees," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 121–142, 2004.

[40] P. Melliès and S. Mimram, "Asynchronous games: Innocence without alternation," in *CONCUR 2007 - Concurrency Theory, 18th International Conference*, ser. Lecture Notes in Computer Science, L. Caires and V. T. Vasconcelos, Eds., vol. 4703. Springer, 2007, pp. 395–411.

[41] S. Castellan, P. Clairambault, S. Rideau, and G. Winskel, "Games and strategies as event structures," *Log. Methods Comput. Sci.*, vol. 13, no. 3, 2017.

[42] V. Danos, "La logique linéaire appliquée à l'étude de divers processus de normalisation (principalement du lambda-calcul)," Ph.D. dissertation, Université Paris 7, France, 1990.

[43] J. Girard, "Locus solum: From the rules of logic to the logic of rules," *Math. Struct. Comput. Sci.*, vol. 11, no. 3, pp. 301–506, 2001.

[44] P. Wadler, "Propositions as sessions," *J. Funct. Program.*, vol. 24, no. 2-3, pp. 384–418, 2014.