ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Eager functions as processes

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Durier, A., Hirschkoff, D., Sangiorgi, D. (2018). Eager functions as processes. N.Y. : Association for Computing Machinery [10.1145/3209108.3209152].

Availability:
This version is available at: https://hdl.handle.net/11585/669217 since: 2019-02-20

Published:
DOI: http://doi.org/10.1145/3209108.3209152

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

12 October 2024

This is the final peer-reviewed accepted manuscript of:

Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. 2018. Eager Functions as Processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (*LICS '18*). Association for Computing Machinery, New York, NY, USA, 364–373.

The final published version is available online at: **https://doi.org/10.1145/3209108.3209152**

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

Eager Functions as Processes

Adrien Durier
Univ. Lyon, ENS de Lyon, CNRS,
UCB Lyon 1, LIP UMR 5668

Daniel Hirschkoff
Univ. Lyon, ENS de Lyon, CNRS,
UCB Lyon 1, LIP UMR 5668

Davide Sangiorgi
Università di Bologna and INRIA

Abstract

We study Milner's encoding of the call-by-value λ-calculus into the π -calculus. We show that, by tuning the encoding to two subcalculi of the π -calculus (Internal π and Asynchronous Local π), the equiv-alence on λ-terms induced by the encoding coincides with Lassen's eager normal-form bisimilarity, extended to handle η-equality. As behavioural equivalence in the π -calculus we consider contextual equivalence and barbed congruence. We also extend the results to preorders.

A crucial technical ingredient in the proofs is the recently-introduced technique of unique solutions of equations, further developed in this paper. In this respect, the paper also intends to be an extended case study on the applicability and expressiveness of the technique.

Keywords pi-calculus, lambda-calculus, full abstraction, call-by-value

Introduction

Milner's work on functions as processes [17, 18], that shows how the evaluation strategies of *call-by-name λ-calculus* and *call-by-value λ-calculus* [1, 21] can be faithfully mimicked in the π -calculus, is generally considered a landmark in Concurrency Theory, and more generally in Programming Language Theory. The comparison with the λ-calculus is a significant expressiveness test for the π -calculus. More than that, it promotes the π -calculus to be a basis for general-purpose programming languages in which communication is the fundamental computing primitive. From the λ-calculus point of view, the comparison provides the means to study λ-terms in contexts other than purely sequential ones, and with the instruments available to reason about processes. Further, Milner's work, and the works that followed it, have contributed to understanding and developing the theory of the π -calculus.

More precisely, Milner shows the operational correspondence between reductions in the λ-terms and in the encoding π -terms. He then uses the correspondence to prove that the encodings are *sound*, i.e., if the processes encoding two λ-terms are behaviourally equivalent, then the source λ-terms are also behaviourally equivalent in the λ-calculus. Milner also shows that the converse, *completeness*, fails, intuitively because the encodings allow one to test the λ-terms in all contexts of the π -calculus — more diverse than those of the λ-calculus.

The main problem that Milner work left open is the characterisation of the equivalence on λ-terms induced by the encoding, whereby two λ-terms are equal if their encodings are behaviourally equivalent π -calculus terms. The question is largely independent of the precise form of behavioural equivalence adopted in the π-calculus because the encodings are deterministic (or at least confluent). In the paper we consider contextual equivalence (that coincides with may testing and trace equivalence) and barbed congruence (that coincides with bisimilarity).

For the call-by-name λ-calculus, the answer was found shortly later [24, 26]: the equality induced is the equality of Lévy-Longo Trees [15], the lazy variant of Böhm Trees. It is actually also possible to obtain Böhm Trees, by modifying the call-by-name encoding so to allow also reductions underneath a λ-abstraction, and by including divergence among the observables [29]. These results show that, at least for call-by-name, the π-calculus encoding, while not fully abstract for the contextual equivalence of the λ-calculus, is in remarkable agreement with the theory of the λ-calculus: several well-known models of the λ-calculus yield Lévy-Longo Trees or Böhm Trees as their induced equivalence [4, 14, 15].

For call-by-value, in contrast, the problem of identifying the equivalence induced by the encoding has remained open, for two main reasons. First, tree structures in call-by-value are less studied and less established than in call-by-name. Secondly, proving completeness of an encoding of λ into π requires sophisticated proof techniques. For call-by-name, for instance, a central role is played by *bisimulation up-to contexts*. For call-by-value, however, existing proof techniques, including 'up-to contexts', appeared not to be powerful enough.

In this paper we study the above open problem for call-by-value. Our main result is that the equivalence induced on λ-terms by their call-by-value encoding into the π-calculus is *eager normal-form bisimilarity* [12, 13]. This is a tree structure for call-by-value, proposed by Lassen as the call-by-value counterpart of Lévy-Longo Trees. Precisely we obtain the variant that is insensitive to η-expansion, called *η-eager normal-form bisimilarity*.

To obtain the results we have however to make a few adjustments to Milner's encoding and/or specialise the target language of the encoding. These adjustments have to do with the presence of free outputs (outputs of known names) in the encoding. We show in the paper that this brings problems when analysing λ-terms with free variables: desirable call-by-value equalities fail. An example is given by the law:

$$I(xV) = xV \tag{1}$$

where I is $\lambda z.\, z$ and V is a value. Two possible solutions are:

1. rule out the free outputs; this essentially means transplanting the encoding onto the Internal π-calculus [25], a version of the π-calculus in which any name emitted in an output is fresh;
2. control the use of capabilities in the π-calculus; for instance taking Asynchronous Local π [16] as the target of the translation. (Controlling capabilities allows one to impose a directionality on names, which, under certain technical conditions, may hide the identity of the emitted names.)

In the paper we consider both approaches, and show that in both cases, the equivalence induced coincides with η-eager normal-form bisimilarity.

In summary, there are two contributions in the paper:

1. Showing that Milner's encoding fails to equate terms that should be equal in call-by-value.
2. Rectifying the encoding, by considering different target calculi, and investigating Milner's problem in such a setting.

The rectification we make does not really change the essence of the encoding – in one case, the encoding actually remains the same. Moreover, the languages used are well-known dialects of the π-calculus, studied in the literature for other reasons. In the encoding, they allow us to avoid certain accidental misuses of the names emitted in the communications. The calculi were not known at the time of Milner's paper [18].

A key role in the completeness proof is played by a technique of *unique solution of equations*, recently proposed [7]. The structure induced by Milner's call-by-value encoding was expected to look like Lassen's trees; however existing proof techniques did not seem powerful enough to prove it. The unique solution technique allows one to derive process bisimilarities from equations whose infinite unfolding does not introduce divergences, by proving that the processes are solutions of the same equations. The technique can be generalised to possibly-infinite systems of equations, and can be strengthened by allowing certain kinds of divergences in equations. In this respect, another goal of the paper is to carry out an extended case study on the applicability and expressiveness of the techniques. Then, a by-product of the study are a few further developments of the technique. In particular, one such result allows us to transplant uniqueness of solutions from a system of equations, for which divergences are easy to analyse, to another one. Another result is about the application of the technique to preorders.

Finally, we consider preorders — thus referring to the preorder on λ-terms induced by a behavioural preorder on their π-calculus encodings. We introduce a preorder on Lassen's trees (preorders had not been considered by Lassen) and show that this is the preorder on λ-terms induced by the call-by-value encoding, when the behavioural relation on π-calculus terms is the ordinary contextual preorder (again, with the caveat of points (1) and (2) above). With the move from equivalences to preorders, the overall structure of the proofs of our full abstraction results remains the same. However, the impact on the application of the unique-solution technique is substantial, because the phrasing of this technique in the cases of preorders and of equivalences is quite different.

Further related work. The standard behavioural equivalence in the λ-calculus is contextual equivalence. Encodings into the π-calculus (be it for call-by-name or call-by-value) break contextual equivalence because π-calculus contexts are richer than those in the (pure) λ-calculus. In the paper we try to understand how far beyond contextual equivalence the discriminating power of the π-calculus brings us, for call-by-value. The opposite approach is to restrict the set of 'legal' π-contexts so to remain faithful to contextual equivalence. This approach has been followed, for call-by-name, and using type systems, in [5, 31].

Open call-by-value has been studied in [3], where the focus is on operational properties of λ-terms; behavioural equivalences are not considered. An extensive presentation of call-by-value, including denotational models, is Ronchi della Rocca and Paolini's book [22].

In [7], the unique-solution technique is used in the completeness proof for Milner's call-by-name encoding. That proof essentially revisits the proof of [26], which is based on bisimulation up-to context. We have explained above that the case for call-by-value is quite different.

Structure of the paper. We recall basic definitions about the call-by-value λ-calculus and the π-calculus in Section 1. The technique of unique solution of equations is introduced in Section 2, together with some new developments. Section 3 presents our analysis of Milner's encoding, beginning with the shortcomings related to the presence of free outputs. The first solution to these shortcomings is to move to the Internal π-calculus: this is described in Section 4. For the proof of completeness, in Section 4.2, we rely on unique solution of equations; we also compare such technique with the 'up-to techniques'. The second solution is to move to the Asynchronous Local π-calculus: this is discussed in Section 5. We show in Section 6 how our results can be adapted to preorders and to contextual equivalence. Finally in Section 7 we highlight conclusions and possible future work.

1 Background material

Throughout the paper, $\mathcal{R}$ ranges over relations. The composition of two relations $\mathcal{R}$ and $\mathcal{R}'$ is written $\mathcal{R}\mathcal{R}'$. We often use infix notation for relations; thus $P\ \mathcal{R}\ Q$ means $(P, Q) \in \mathcal{R}$. A tilde represents a tuple. The i-th element of a tuple $\widetilde{P}$ is referred to as P_i. Our notations are extended to tuples componentwise. Thus $\widetilde{P}\ \mathcal{R}\ \widetilde{Q}$ means $P_i\ \mathcal{R}\ Q_i$ for all components.

1.1 The call-by-value 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 [4, 8]. Here and in the rest of the paper (including when reasoning about π processes), we adopt the usual "Barendregt convention". This will allow us to assume freshness of bound variables and names whenever needed. The set of free variables in the term M is $\mathsf{fv}(M)$. 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 $\lambda\widetilde{x}.M$ if the length of $\widetilde{x}$ is not important. Symbol Ω stands for the always-divergent term $(\lambda x.xx)(\lambda x.xx)$.

A *context* is a term with a hole $[\cdot]$, possibly occurring more than once. If C is a context, $C[M]$ is a shorthand for C where the hole $[\cdot]$ is substituted by M. An *evaluation context* is a special kind of context, with exactly one hole $[\cdot]$, and in which the inserted term can immediately run. In the pure λ-calculus *values* are abstractions and variables.

$$\begin{array}{rrcl} \text{Evaluation contexts} & C_{\mathrm{e}} & := & [\cdot] \mid C_{\mathrm{e}}M \mid VC_{\mathrm{e}} \\ \text{Values} & V & := & x \mid \lambda x.M \end{array}$$

In call-by-value, substitutions replace variables with values; we call them *value substitutions*.

Eager reduction (or β_{v}-reduction), $\longrightarrow \subseteq \Lambda \times \Lambda$, is determined by the rule:

$$C_{\mathrm{e}}[(\lambda x.M)V] \longrightarrow C_{\mathrm{e}}[M\{V/x\}] \ .$$

We write $\Longrightarrow$ for the reflexive transitive closure of $\longrightarrow$. A term in *eager normal form* is a term that has no eager reduction.

Proposition 1.1. *1. If $M \longrightarrow M'$, then $C_e[M] \longrightarrow C_e[M']$ and $M\sigma \longrightarrow M'\sigma$, for any value substitution σ.*
2. Terms in eager normal form are either values or of the shape $C_e[xV]$.

Therefore, given a term M, either $M \Longrightarrow M'$ where M' is a term in eager normal form, or there is an infinite reduction sequence starting from M. In the first case, M *has eager normal form* M', written $M \Downarrow M'$, in the second M *diverges*, written $M \Uparrow$. We write $M \Downarrow$ when $M \Downarrow M'$ for some M'.

Definition 1.2 (Contextual equivalence)**.** Given $M,\ N \in \Lambda$, we say that M and N are contextually equivalent, written $M \simeq^{\Lambda}_{\mathrm{ct}} N$, if for any context C, we have $C[M] \Downarrow$ iff $C[N] \Downarrow$.

1.2 Tree semantics for call-by-value

We recall *eager normal-form bisimilarity* [12, 13, 30].

Definition 1.3 (Eager normal-form bisimulation)**.** A relation $\mathcal{R}$ between λ-terms is an *eager normal-form bisimulation* if, whenever $M \mathcal{R} N$, one of the following holds:

1. both M and N diverge;
2. $M \Downarrow C_e[xV]$ and $N \Downarrow C'_e[xV']$ for some x, values V, V', and evaluation contexts C_e and C'_e with $V \mathcal{R} V'$ and $C_e[z] \mathcal{R} C'_e[z]$ for a fresh z;
3. $M \Downarrow \lambda x.M'$ and $N \Downarrow \lambda x.N'$ for some x, M', N' with $M' \mathcal{R} N'$;
4. $M \Downarrow x$ and $N \Downarrow x$ for some x.

Eager normal-form bisimilarity, $\leftrightarrows$, is the largest eager normal-form bisimulation.

Essentially, the structure of a λ-term that is unveiled by Definition 1.3 is that of a (possibly infinite) tree obtained by repeatedly applying β_v-reduction, and branching a tree whenever instantiation of a variable is needed to continue the reduction (clause (2)). We call such trees *Eager Trees* (ETs) and accordingly also call eager normal-form bisimilarity the *Eager-Tree equality*.

Example 1.4. Relation $\leftrightarrows$ is strictly finer than contextual equivalence $\simeq^{\Lambda}_{\mathrm{ct}}$: the inclusion $\leftrightarrows \subseteq \simeq^{\Lambda}_{\mathrm{ct}}$ follows from the congruence properties of $\leftrightarrows$ [12]; for the strictness, examples are the following equalities, that hold for $\simeq^{\Lambda}_{\mathrm{ct}}$ but not for $\leftrightarrows$:

$$\Omega = (\lambda y.\,\Omega)(xV) \qquad xV = (\lambda y.\,xV)(xV)\ .$$

Example 1.5 (η rule)**.** The η-rule is not valid for $\leftrightarrows$. For instance, we have $\Omega \not\leftrightarrows \lambda x.\,\Omega x$. The rule is not even valid on values, as we also have $\lambda y.\,xy \not\leftrightarrows x$. It holds however for abstractions: $\lambda y.\,(\lambda x.\,M)y \leftrightarrows \lambda x.\,M$ when $y \notin \mathrm{fv}(M)$.

The failure of the η-rule $\lambda y.\,xy \not\leftrightarrows x$ is troublesome as, under any closed value substitution, the two terms are indeed eager normal-form bisimilar (as well as contextually equivalent). Thus *η-eager normal-form bisimilarity* [12] takes η-expansion into account so to recover such missing equalities.

Definition 1.6 (η-eager normal-form bisimulation)**.** A relation $\mathcal{R}$ between λ-terms is an *η-eager normal-form bisimulation* if, whenever $M \mathcal{R} N$, either one of the clauses of Definition 1.3, or one of the two following additional clauses, hold:

5. $M \Downarrow x$ and $N \Downarrow \lambda y.\,N'$ for some x, y, and N' such that $N' \Downarrow C_e[xV]$, with $y \mathcal{R} V$ and $z \mathcal{R} C_e[z]$ for some value V, evaluation context C_e, and fresh z.
6. the converse of (5), i.e., $N \Downarrow x$ and $M \Downarrow \lambda y.\,M'$ for some x, y, and M' such that $M' \Downarrow C_e[xV]$, with $V \mathcal{R} y$ and $C_e[z] \mathcal{R} z$ for some value V, evaluation context C_e, and fresh z.

Then *η-eager normal-form bisimilarity*, $\leftrightarrows_\eta$, is the largest η-eager normal-form bisimulation.

We sometimes call relation $\leftrightarrows_\eta$ the *η-Eager-Tree equality.*

Remark 1.7. *Definition 1.6 coinductively allows η-expansions to occur underneath other η-expansions, hence trees with infinite η-expansions may be equated with finite trees. For instance,*

$$x \leftrightarrows_\eta \lambda y.\,xy \leftrightarrows_\eta \lambda y.\,x(\lambda z.\,yz) \leftrightarrows_\eta \lambda y.\,x(\lambda z.\,y(\lambda w.\,zw)) \leftrightarrows_\eta \ldots$$

A concrete example is given by taking a fixpoint Y, and setting $f \stackrel{def}{=} (\lambda zxy.\,x(zy))$. We then have $Yfx \Longrightarrow \lambda y.\,x(Yfy)$, and then $x(Yfy) \Longrightarrow x(\lambda z.\,y(Yfz))$, and so on. Hence, we have $x \leftrightarrows_\eta Yfx$.

1.3 The π-calculus, Iπ and ALπ

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 $F\langle\widetilde{a}\rangle$. We use P, Q for processes, F for abstractions. Processes and abstractions form the set of *π-agents* (or simply *agents*), ranged over by A. Small letters $a, b, \ldots, x, y, \ldots$ range over the infinite set of names. The grammar of the π-calculus is thus:

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

In prefixes $a(\widetilde{b})$ and $\overline{a}\langle\widetilde{b}\rangle$, we call a the *subject* and $\widetilde{b}$ the *object*. When the tilde is empty, the surrounding brackets in prefixes will be omitted. We often abbreviate $\boldsymbol{\nu} a\,\boldsymbol{\nu} b\,P$ as $(\boldsymbol{\nu} a, b)P$. An input prefix $a(\widetilde{b}).\,P$, a restriction $\boldsymbol{\nu} b\,P$, and an abstraction $(\widetilde{b})\,P$ are binders for names $\widetilde{b}$ and b, respectively, and give rise in the expected way to the definition of *free names* (fn) and *bound names* (bn) of a term or a prefix, and α-conversion. An agent is *name-closed* if it does not contain free names. As in the λ-calculus, following the usual Barendregt convention we identify processes or actions which only differ on the choice of the bound names. The symbol = will mean "syntactic identity modulo α-conversion". Sometimes, we use $\stackrel{\text{def}}{=}$ as abbreviation mechanism, to assign a name to an expression to which we want to refer later.

We use constants, ranged over by K for writing recursive definitions. Each constant has a defining equation of the form $K \stackrel{\triangle}{=} (\widetilde{x})\,P$, 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).

Since the calculus is polyadic, we assume a *sorting system* [19] to avoid disagreements in the arities of the tuples of names carried by a given name and in applications of abstractions. We will not present the sorting system because it is not essential. The reader should take for granted that all agents described obey a sorting. A *context* C of π is a π-agent in which some subterms have been replaced by the hole $[\cdot]$ or, if the context is polyadic, with indexed

holes $[\cdot]_1, \ldots, [\cdot]_n$; then $C[A]$ or $C[\widetilde{A}]$ is the agent resulting from replacing the holes with the terms A or $\widetilde{A}$.

We omit the operators of sum and matching (not needed in the encodings). We refer to [19] for detailed discussions on the operators of the language. We assign parallel composition the lowest precedence among the operators.

Operational semantics. The operational semantics of the π-calculus is standard [28] (including the labelled transition system). The reference behavioural equivalence for π-calculi will be the usual *barbed congruence*. We recall its definition, on a generic subset $\mathcal{L}$ of π-calculus processes. A $\mathcal{L}$*-context* is a process of $\mathcal{L}$ with a single hole $[\cdot]$ in it (the hole has a sort too, as it could be in place of an abstraction). We write $P \Downarrow_a$ if P can make an output action whose subject is a, possibly after some internal moves. (We make only output observable because this is standard in asynchronous calculi; adding also observability of inputs does not affect barbed congruence on the synchronous calculi we will consider.)

Definition 1.8 (Barbed c ongruence)**.** *B arbed b isimilarity* i s the largest symmetric relation $\dot{\simeq}$ on π-calculus processes such that $P \dot{\simeq} Q$ implies:

1. If $P \Longrightarrow P'$ then there is Q' such that $Q \Longrightarrow Q'$ and $P' \dot{\simeq} Q'$.
2. $P \Downarrow_a$ iff $Q \Downarrow_a$.

Let $\mathcal{L}$ be a set of π-calculus agents, and $A, B \in \mathcal{L}$. We say that A and B are *barbed congruent in* $\mathcal{L}$, written $A \simeq^{\mathcal{L}} B$, if for each (well-sorted) $\mathcal{L}$-context C, it holds that $C[A] \dot{\simeq} C[B]$.

Remark 1.9. *Barbed congruence has been uniformly defined on processes and abstractions (via a quantification on all process contexts). Usually, however, definitions will only be given for processes; it is then intended that they are extended to abstractions by requiring closure under ground parameters, i.e., by supplying fresh names as arguments.*

As for all contextually-defined behavioural relations, so barbed congruence is hard to work with. In all calculi we consider, it can be characterised in terms of *ground bisimilarity*, under the (mild) condition that the processes are image-finite up to $\approx$. (We recall that the class of processes *image-finite up to* $\approx$ is the largest subset $\mathcal{IF}$ of π-calculus processes which is derivation closed and such that $P \in \mathcal{IF}$ implies that, for all actions μ, the set $\{P' \mid P \stackrel{\mu}{\Longrightarrow} P'\}$ quotiented by $\approx$ is finite. The definition is extended to abstractions as by Remark 1.9.) All the agents in the paper, including those obtained by encodings of the λ-calculus, are image-finite up to $\approx$. The distinctive feature of *ground* bisimilarity is that it does not involve instantiation of the bound names of inputs (other than by means of fresh names), and similarly for abstractions. In the remainder, we omit the adjective 'ground'.

Definition 1.10 (Bisimilarity)**.** A symmetric relation $\mathcal{R}$ on π-processes is a *bisimulation*, if whenever $P \mathcal{R} Q$ and $P \xrightarrow{\mu} P'$, then $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ 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 $\mathcal{R}$.

We will use two subcalculi: the Internal π-calculus (Iπ), and the Asynchronous Local π-calculus (ALπ), obtained by placing certain constraints on prefixes.

Iπ. In Iπ, all outputs are bound. This is syntactically enforced by replacing the output construct with the bound-output construct $\overline{a}(\widetilde{b}).P$, which, with respect to the grammar of the ordinary π-calculus, is an abbreviation for $\nu\widetilde{b}\overline{a}\langle\widetilde{b}\rangle.P$. In all tuples (input, output, abstractions, applications) the components are pairwise distinct so to make sure that distinctions among names are preserved by reduction.

ALπ. ALπ is defined by enforcing that in an input $a(\widetilde{b}).P$, all names in $\widetilde{b}$ appear only in output position in P. Moreover, ALπ being *asynchronous*, output prefixes have no continuation; in the grammar of the π-calculus this corresponds to having only outputs of the form $\overline{a}\langle\widetilde{b}\rangle.\mathbf{0}$ (which we will simply write $\overline{a}\langle\widetilde{b}\rangle$). In AL$\pi$, to maintain the characterisation of barbed congruence as (ground) bisimilarity, the transition system has to be modified [16], allowing the dynamic introduction of additional processes (the 'links', sometimes also called forwarders).

Theorem 1.11.
1. *In Iπ, on agents that are image-finite up to $\approx$, barbed congruence and bisimilarity coincide.*
2. *In ALπ, on agents that are image-finite up to $\approx$ and where no free name is used in input, barbed congruence and bisimilarity coincide.*

All encodings of the λ-calculus (into Iπ and ALπ) in the paper satisfy the conditions of Theorem 1.11. Thus we will be able to use bisimilarity as a proof technique for barbed congruence. (In part (2) of the theorem, the condition on inputs can be removed by adopting an asynchronous variant of bisimilarity; however, the synchronous version is easier to use in our proofs based on unique solution of equations).

2 Unique solutions in Iπ and ALπ

We adapt the proof technique of unique solution of equations, from [7] to the calculi Iπ and ALπ, in order to derive bisimilarity results. The technique is discussed in [7] on the asynchronous π-calculus (for possibly-infinite systems of equations). The structure of the proofs for Iπ and ALπ is similar; in particular the completeness part is essentially the same because bisimilarity is the same. The differences in the syntax of Iπ, and in the transition system of ALπ, show up only in certain technical details of the soundness proofs.

We need variables to write equations. We use capital letters X, Y, Z for these variables and call them *equation variables*. The body of an equation is a name-closed abstraction possibly containing equation variables (that is, applications can also be of the form $X\langle\overline{a}\rangle$). We use E to range over such expressions; and $\mathcal{E}$ to range over systems of equations, defined as follows. In the definitions below, the indexing set I can be infinite.

Definition 2.1. Assume that, for each i of a countable indexing set I, we have a variable X_i, and an expression E_i, possibly containing some 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; we sometimes use X_i to refer to that equation.)

A system of equations is *guarded* if each occurrence of a variable in the body of an equation is underneath a prefix.

$E[\widetilde{F}]$ is the abstraction resulting from E by replacing each variable X_i with the abstraction F_i (as usual assuming $\widetilde{F}$ and $\widetilde{X}$ have the same sort).

Definition 2.2. 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 2.3 (Syntactic solutions)**.** The syntactic solutions of the system of equations $\widetilde{X} = \widetilde{E}$ are the recursively defined constants $\mathrm{K}_{\widetilde{E},i} \triangleq E_i[\widetilde{\mathrm{K}}_{\widetilde{E}}]$, for each $i \in I$, where I is the indexing set of the system.

The syntactic solutions of a system of equations are indeed solutions of it.

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, 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 $\left(P_i \xrightarrow{\mu_i} P_{i+1}\right)_i$. In the case of an abstraction, F has a divergence if the process $F\langle\widetilde{a}\rangle$ has a divergence, where $\widetilde{a}$ are fresh names. A tuple of agents $\widetilde{A}$ *is divergence-free* if none of the components A_i has a divergence.

The following result is the technique we rely on to establish completeness of the encoding. As announced above, it holds in both Iπ and ALπ.

Theorem 2.4. *In Iπ and ALπ, a guarded system of equations with divergence-free syntactic solutions has unique solution for $\approx$.*

Techniques for ensuring termination, hence divergence freedom, for the π-calculus have been studied in, e.g., [6, 27, 32].

2.1 Further Developments

We present some further developments to the theory of unique solution of equations, that are needed for the results in this paper. The first result allows us to derive the unique-solution property for a system of equations from the analogous property of an extended system.

Definition 2.5. A system of equations $\mathcal{E}'$ *extends* system $\mathcal{E}$ if there exists a fixed set of indices J such that any solution of $\mathcal{E}$ can be obtained from a solution of $\mathcal{E}'$ by removing the components corresponding to indices in J.

Theorem 2.6. *Consider two systems of equations $\mathcal{E}'$ and $\mathcal{E}$ where $\mathcal{E}'$ extends $\mathcal{E}$. If $\mathcal{E}'$ has a unique solution, then the property also holds for $\mathcal{E}$.*

We shall use Theorem 2.6 in Section 4.2, in a situation where we transform a certain system into another one, whose uniqueness of solutions is easier to establish.

Remark 2.7. *We cannot derive Theorem 2.6 by comparing the syntactic solutions of the two systems $\mathcal{E}'$ and $\mathcal{E}$. For instance, the equations $X = \tau.X$ and $X = \tau.\tau.\tau\dots$ have (strongly) bisimilar syntactic solutions, yet only the latter equation has the unique-solution property. (Further, Theorem 2.6 allows us to compare systems of different size.)*

The second development is a generalisation of Theorem 2.4 to preorders; we postpone its presentation to Section 6.

3 Milner's encodings

3.1 Background

Milner noticed [17, 18] that his call-by-value encoding can be easily tuned so to mimic forms of evaluation in which, in an application MN, the function M is run first, or the argument N is run first, or function and argument are run in parallel (the proofs are actually carried out for this last option). We chose here the first one, because it is more in line with ordinary call-by-value. A discussion on the 'parallel' call-by-value is deferred to Section 7.

The core of any encoding of the λ-calculus into a process calculus is the translation of function application. This becomes a particular form of parallel combination of two processes, the function and its argument; β_{v}-reduction is then modeled as process interaction.

The encoding of a λ-term is parametric over a name; this may be thought of as the *location* of that term, or as its *continuation*. A term that becomes a value signals so at its continuation name and, in doing so, it grants access to the body of the value. Such body is replicated, so that the value may be copied several times. When the value is a function, its body can receive two names: (the access to) its value-argument, and the following continuation. In the translation of application, first the function is run, then the argument; finally the function is informed of its argument and continuation.

In the original paper [17], Milner presented two candidates for the encoding of call-by-value λ-calculus [21]. They follow the same idea of translation, but with a technical difference in the rule for variables. One encoding, $\mathcal{V}$, is so defined:

$$\begin{aligned}
\mathcal{V}[\![\lambda x.M]\!] &\stackrel{\text{def}}{=} (p)\,\overline{p}(y).\,!y(x,q).\,\mathcal{V}[\![M]\!]\langle q\rangle \\
\mathcal{V}[\![MN]\!] &\stackrel{\text{def}}{=} \\
&(p)\,(\boldsymbol{\nu} q\,)(\mathcal{V}[\![M]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\,(\mathcal{V}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{y}\langle w,p\rangle)) \\
\mathcal{V}[\![x]\!] &\stackrel{\text{def}}{=} (p)\,\overline{p}\langle x\rangle
\end{aligned}$$

In the other encoding, $\mathcal{V}'$, application and λ-abstraction are treated as in $\mathcal{V}$; the rule for variables is:

$$\mathcal{V}'[\![x]\!] \stackrel{\text{def}}{=} (p)\,\overline{p}(y).\,!y(z,q).\,\overline{x}\langle z,q\rangle \ .$$

The encoding $\mathcal{V}$ is more efficient than $\mathcal{V}'$, as it uses fewer communications.

3.2 Some problems with the encoding

The immediate free output in the encoding of variables in $\mathcal{V}$ breaks the validity of β_{v}-reduction; i.e., there exist a term M and a value V such that $\mathcal{V}[\![(\lambda x.M)V]\!] \not\approx \mathcal{V}[\![M\{^V/x\}]\!]$ [23]. The encoding $\mathcal{V}'$ fixes this by communicating, instead of a free name, a fresh pointer to that name. Technically, the initial free output of x is replaced by a bound output coupled with a link to x (the process $!y(z,q).\,\overline{x}\langle z,q\rangle$, receiving at y and re-emitting at x). Thus β_{v}-reduction is validated [23]. (The final version of Milner's paper [18], was written after the results in [23] were known and presents only the encoding $\mathcal{V}'$.)

Nevertheless, $\mathcal{V}'$ only delays the free output, as the added link contains itself a free output. As a consequence, we can show that other desirable equalities of call-by-value are broken. An example is law (1) from the Introduction, as stated by Proposition 3.1 below. This law is desirable (and indeed valid for contextual equivalence, or the Eager-Tree equality) intuitively because, in any substitution closure of the law, either both terms diverge, or they converge

$$
\begin{aligned}
\mathcal{I}[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, !y(x,q).\, \mathcal{I}[\![M]\!]\langle q\rangle \\
\mathcal{I}[\![x]\!] &\stackrel{\text{def}}{=} (p)\ \overline{p}(y).\, y \triangleright x \\
\mathcal{I}[\![MN]\!] &\stackrel{\text{def}}{=} (p)\ \boldsymbol{\nu} q\, \big(\mathcal{I}[\![M]\!]\langle q\rangle \mid q(y).\, \boldsymbol{\nu} r\, \big(\mathcal{I}[\![N]\!]\langle r\rangle \mid \\
&\qquad\qquad r(w).\, \overline{y}(w',p').\, (w' \triangleright w \mid p' \triangleright p)\big)\big)
\end{aligned}
$$

Figure 1. The encoding into Iπ

to the same value. The same argument holds for their λ-closures, $\lambda x.\, xV$ and $\lambda x.\, I(xV)$. We recall that $\simeq^{\pi}$ is barbed congruence in the π-calculus.

Proposition 3.1. *For any value V, we have:*

$$\mathcal{V}'[\![I(xV)]\!] \not\simeq^{\pi} \mathcal{V}'[\![xV]\!] \text{ and } \mathcal{V}[\![I(xV)]\!] \not\simeq^{\pi} \mathcal{V}[\![xV]\!] \ .$$

(The law is violated also under coarser equivalences, such as contextual equivalence.) Technically, the reason why the law fails in π can be illustrated when $V = y$, for encoding $\mathcal{V}$. We have:

$$
\begin{aligned}
\mathcal{V}[\![xy]\!]\langle p\rangle &\simeq^{\pi} \overline{x}(v).\, \boldsymbol{\nu} w \qquad (\overline{v}\langle w,p\rangle \mid !w(u).\, \overline{y}\langle u\rangle) \\
\mathcal{V}[\![I(xy)]\!]\langle p\rangle &\simeq^{\pi} \overline{x}(v).\, (\boldsymbol{\nu} w, q)(\overline{v}\langle w,q\rangle \mid !w(u).\, \overline{y}\langle u\rangle \\
&\qquad\qquad \mid q(z).\, \overline{p}(z').\, !z'(w').\, \overline{z}\langle w'\rangle)
\end{aligned}
$$

In presence of the normal form xy, the identity I becomes observable. Indeed, in the second term, a fresh name, q, is sent instead of continuation p, and a link between q and p is installed. This corresponds to a law which is valid in ALπ, but not in π.

This problem can be avoided by iterating the transformation that takes us from $\mathcal{V}$ to $\mathcal{V}'$ (i.e., the replacement of a free output with a bound output so to avoid all emissions of free names). Thus the target language becomes Internal π; the resulting encoding is analysed in Section 4.

Another solution is to control the use of name capabilities in processes. In this case the target language becomes ALπ, and we need not modify the initial encoding $\mathcal{V}$. This situation is analysed in Section 5.

Moreover, in both solutions, the use of link processes validates the following law — a form of η-expansion — (the law fails for Milner's encoding into the π-calculus):

$$\lambda y.\, xy = x$$

In the call-by-value λ-calculus this is a useful law (that holds because substitutions replace variables with values).

4 Encoding in the Internal π-calculus

4.1 Encoding and soundness

Figure 1 presents the encoding into Iπ, derived from Milner's encoding by removing the free outputs as explained in Section 3. Process $a \triangleright b$ represents a *link* (sometimes called forwarder; for readability we have adopted the infix notation $a \triangleright b$ for the constant $\triangleright$). It transforms all outputs at a into outputs at b (therefore a, b are names of the same sort). Thus the body of $a \triangleright b$ is replicated, unless a and b are *continuation names* (names such as p, q, r over which the encoding of a term is abstracted). The definition of the constant $\triangleright$ therefore is:

$$
\triangleright \ \triangleq \ \begin{cases} (p,q)\ p(x).\, \overline{q}(y).\, y \triangleright x & \\ \qquad \text{if } p,q \text{ are continuation names} & \\ (x,y)\ !x(p,z).\, \overline{y}(q,w).\, (q \triangleright p \mid w \triangleright z) & \\ \qquad\qquad\qquad\qquad \text{otherwise} & \end{cases}
$$

(The distinction between continuation names and the other sorts of names is not necessary, but simplifies the proofs.)

The encoding validates β_{v}-reduction.

Lemma 4.1 (Validity of β_{v}-reduction)**.** *For any M, N in Λ, $M \longrightarrow N$ implies $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$.*

The structure of the proof of soundness of the encoding is similar to that for the analogous property for Milner's call-by-name encoding with respect to Levy-Longo Trees [26]. The details are however different, as in call-by-value both the encoding and the trees (the Eager Trees extended to handle η-expansion) are more complex.

We first need to establish an operational correspondence for the encoding. For this we make use of an optimised encoding, obtained from the one in Figure 1 by performing a few (deterministic) reductions, at the price of a more complex definition. Precisely, in the encoding of application, we remove some of the initial communications, including those with which a term signals that it has become a value. Correctness of the optimisations is established by algebraic reasoning.

Using the operational correspondence, we then show that the observables for bisimilarity in the encoding π-terms imply the observables for η-eager normal-form bisimilarity in the encoded λ-terms. The delicate cases are those in which a branch in the tree of the terms is produced — case (2) of Definition 1.3 — and where an η-expansion occurs — thus a variable is equivalent to an abstraction, cases (5) and (6) of Definition 1.6.

For the branching, we exploit a decomposition property on π-terms, roughly allowing us to derive from the bisimilarity of two parallel compositions the componentwise bisimilarity of the single components. For the η-expansion, if $\mathcal{I}[\![x]\!] \approx \mathcal{I}[\![\lambda z.\, M]\!]$, where $M \Downarrow C_{\mathrm{e}}[xV]$, we use a coinductive argument to derive $V \leftrightarrows_{\eta} z$ and $C_{\mathrm{e}}[y] \leftrightarrows_{\eta} y$, for y fresh; from this we then obtain $\lambda z.\, M \leftrightarrows_{\eta} x$.

Lemma 4.2 (Soundness)**.** *For any $M, N \in \Lambda$, if $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$ then $M \leftrightarrows_{\eta} N$.*

4.2 Completeness and Full Abstraction

To ease the reader into the proof, we first show the completeness for $\leftrightarrows$, rather than $\leftrightarrows_{\eta}$.

The system of equations. Suppose $\mathcal{R}$ is an eager normal-form bisimulation. 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 $\mathcal{R}$. We then use Theorem 2.4 and Theorem 2.6 to show that $\mathcal{E}_{\mathcal{R}}$ has a unique solution.

We assume an ordering on names and variables, so to be able to view (finite) sets of these as tuples. Moreover, if F is an abstraction, say $(\widetilde{a})\ P$, then $(\widetilde{y})\ F$ is an abbreviation for its uncurrying $(\widetilde{y}, \widetilde{a})\ P$.

There is one equation $X_{M,N} = E_{M,N}$ for each pair $(M, N) \in \mathcal{R}$. The body $E_{M,N}$ is essentially the encoding of the eager normal form of M and N, with the variables of the equations representing the coinductive hypothesis. To formalise this, we extend the encoding

of the λ-calculus to equation variables by setting

$$\mathcal{I}[\![X_{M,N}]\!] \stackrel{\text{def}}{=} (p)\ X_{M,N}\langle \widetilde{y}, p\rangle \qquad \text{where } \widetilde{y} = \mathsf{fv}(M,N)\ .$$

We now describe the equation $X_{M,N} = E_{M,N}$, for $(M,N) \in \mathcal{R}$. The equation is parametrised on the free variables of M and N (to ensure that the body $E_{M,N}$ is a name-closed abstraction) and an additional continuation name (as all encodings of terms). Below $\widetilde{y} = \mathsf{fv}(M,N)$.

1. If $M \Downarrow x$ and $N \Downarrow x$, then the equation is the encoding of x:
$$\begin{aligned} X_{M,N} &= (\widetilde{y})\ \mathcal{I}[\![x]\!] \\ &= (\widetilde{y}, p)\ \overline{p}(z).\, z \triangleright x \end{aligned}$$
2. If $M \Uparrow$ and $N \Uparrow$, then the equation uses a purely-divergent term; we choose the encoding of Ω:
$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\Omega]\!]$$
3. If $M \Downarrow \lambda x.\, M'$ and $N \Downarrow \lambda x.\, N'$, then the equation encodes an abstraction whose body refers to the normal forms of M', N', via the variable $X_{M',N'}$:
$$\begin{aligned} X_{M,N} &= (\widetilde{y})\ \mathcal{I}[\![\lambda x.\, X_{M',N'}]\!] \\ &= (\widetilde{y}, p)\ \overline{p}(z).\, !z(x,q).\, X_{M',N'}\langle \widetilde{y}', q\rangle \end{aligned}$$
4. If $M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C'_{\mathrm{e}}[xV']$, we separate the evaluation contexts and the values, as in Definition 1.3. In the body of the equation, this is achieved by: (i) rewriting $C_{\mathrm{e}}[xV]$ into $(\lambda z.\, C_{\mathrm{e}}[z])(xV)$, for some fresh z, and similarly for C'_{e} and V' (such a transformation is valid for $\leftrightarrows$); and (ii) referring to the variable for the evaluation contexts, $X_{C_{\mathrm{e}}[z], C'_{\mathrm{e}}[z]}$, and to the variable for the values, $X_{V,V'}$. This yields the equation (for z fresh):
$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![(\lambda z.\, X_{C_{\mathrm{e}}[z], C'_{\mathrm{e}}[z]})\ (x\ X_{V,V'})]\!]$$

As an example, suppose $(I, \lambda x.\, M) \in \mathcal{R}$, where $I = \lambda x.\, x$ and $M = (\lambda zy.\, z)xx'$. The free variables of M are x and x'. We obtain the following equations:

1. $$\begin{aligned} X_{I,\lambda x.M} &= (x')\ \mathcal{I}[\![\lambda x.\, X_{x,M}]\!] \\ &= (x', p)\ \overline{p}(y).\, !y(x,q).\, X_{x,M}\langle x, x', q\rangle \end{aligned}$$
2. $$\begin{aligned} X_{x,M} &= (x, x')\ \mathcal{I}[\![x]\!] \\ &= (x, x', p)\ \overline{p}(y).\, y \triangleright x \end{aligned}$$

Solutions of $\mathcal{E}_{\mathcal{R}}$. Having set the system of equations for $\mathcal{R}$, we now define solutions for it from the encoding of the pairs in $\mathcal{R}$.

We can view the relation $\mathcal{R}$ as an ordered sequence of pairs (e.g., assuming some lexicographical ordering). Then $\mathcal{R}_i$ indicates the tuple obtained by projecting the pairs in $\mathcal{R}$ onto the i-th component ($i = 1, 2$). Moreover (M_j, N_j) is the j-th pair in $\mathcal{R}$, and $\widetilde{y_j}$ is $\mathsf{fv}(M_j, N_j)$.

We write $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ for the closed abstractions resulting from the encoding of $\mathcal{R}_1$, i.e., the tuple whose j-th component is $(\widetilde{y_j})\ \mathcal{I}[\![M_j]\!]$, and similarly for $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$.

Lemma 4.3. *$\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of $\mathcal{E}_{\mathcal{R}}$.*

Proof. We show that each component of $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ is solution of the corresponding equation, i.e., for the j-th component we show $(\widetilde{y_j})\ \mathcal{I}[\![M_j]\!] \approx E_{M_j,N_j}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]]$.

We reason by cases over the shape of the eager normal form of M_j, N_j. The most interesting case is when $M_j \Downarrow C_{\mathrm{e}}[xV]$, in which case we use the following equality (for z fresh), which is proved using algebraic reasoning:

$$\mathcal{I}[\![(\lambda z.\, C_{\mathrm{e}}[z])(xV)]\!] \approx \mathcal{I}[\![C_{\mathrm{e}}[xV]]\!]\ . \tag{2}$$

We also exploit the validity of β_{v} for $\approx$ (Lemma 4.1). □

Unique solution for $\mathcal{E}_{\mathcal{R}}$. We use Theorem 2.6 to prove uniqueness of solutions for $\mathcal{E}_{\mathcal{R}}$. The only delicate requirement is the one on divergence for the syntactic solution. We introduce for this an auxiliary system of equations, $\mathcal{E}'_{\mathcal{R}}$, that extends $\mathcal{E}_{\mathcal{R}}$, and whose syntactic solutions have no τ-transition and hence trivially satisfy the requirement. Like the original system $\mathcal{E}_{\mathcal{R}}$, so the new one $\mathcal{E}'_{\mathcal{R}}$ is defined by inspection of the pairs in $\mathcal{R}$; in $\mathcal{E}'_{\mathcal{R}}$, however, a pair of $\mathcal{R}$ may sometimes yield more than one equation. Thus, let $(M,N) \in \mathcal{R}$ with $\widetilde{y} = \mathsf{fv}(M,N)$.

1. When $M \Uparrow$ and $N \Uparrow$, the equation is
$$X_{M,N} = (\widetilde{y}, p)\ \mathbf{0}\ .$$
2. When $M \Downarrow V$ and $N \Downarrow V'$, we introduce a new equation variable $X^{\mathcal{V}}_{V,V'}$ and a new equation; this will allow us, in the following step (3), to perform some optimisations. The equation is
$$X_{M,N} = (\widetilde{y}, p)\ \overline{p}(z).\, X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle\ ,$$
and we have, accordingly, the two following additional equations corresponding to the cases where values are functions or variables:
$$\begin{aligned} X^{\mathcal{V}}_{\lambda x.M', \lambda x.N'} &= (z, \widetilde{y})\ !z(x,q).\, X_{M',N'}\langle \widetilde{y}', q\rangle \\ X^{\mathcal{V}}_{x,x} &= (z, x)\ z \triangleright x \end{aligned}$$
3. When $M \Downarrow C_{\mathrm{e}}[xV]$ and $N \Downarrow C_{\mathrm{e}}[xV']$, we refer to $X^{\mathcal{V}}_{V,V'}$, instead of $X_{V,V'}$, so to remove all initial reductions in the corresponding equation for $\mathcal{E}_{\mathcal{R}}$. The first action thus becomes an output:
$$\begin{aligned} &X_{M,N} = \\ &(\widetilde{y}, p)\ \overline{x}(z,q).\, (X^{\mathcal{V}}_{V,V'}\langle z, \widetilde{y}'\rangle \mid q(w).\, X_{C_{\mathrm{e}}[w], C'_{\mathrm{e}}[w]}\langle \widetilde{y}'', p\rangle) \end{aligned}$$

Lemmas 4.4 and 4.5 are needed to apply Theorem 2.6. (In the statement of Lemma 4.4, 'extend' is as by Definition 2.5.)

Lemma 4.4. *The system of equations $\mathcal{E}'_{\mathcal{R}}$ extends the system of equations $\mathcal{E}_{\mathcal{R}}$.*

Proof. The new system $\mathcal{E}'_{\mathcal{R}}$ is obtained from $\mathcal{E}_{\mathcal{R}}$ by modifying the equations and adding new ones. Ones shows that the solutions to the common equations are the same, using algebraic reasoning. □

Lemma 4.5. *$\mathcal{E}'_{\mathcal{R}}$ has a unique solution.*

Proof. Divergence-freedom for the syntactic solutions of $\mathcal{E}'_{\mathcal{R}}$ holds because in the equations each name (bound or free) can appear either only in inputs or only in outputs. As a consequence, since the labelled transition system is ground (names are only replaced by fresh ones), no τ-transition can ever be performed, after any number of visible actions. Further, $\mathcal{E}'_{\mathcal{R}}$ is guarded. Hence we can apply Theorem 2.4. □

Lemma 4.6 (Completeness for $\leftrightarrows$)**.** *$M \leftrightarrows N$ implies $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$, for any $M, N \in \Lambda$.*

Proof. Consider an eager normal-form bisimulation $\mathcal{R}$, and the corresponding systems of equations $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}'_{\mathcal{R}}$. Lemmas 4.5 and 4.4 allow us to apply Theorem 2.6 and deduce that $\mathcal{E}_{\mathcal{R}}$ has a unique solution. By Lemma 4.3, $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of $\mathcal{E}_{\mathcal{R}}$. Thus, from $M \ \mathcal{R} \ N$, we deduce $(\widetilde{y})\ \mathcal{I}[\![M]\!] \approx (\widetilde{y})\ \mathcal{I}[\![N]\!]$, where $\widetilde{y} = \mathrm{fv}(M, N)$. Hence also $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$.

□

Completeness for $\leftrightarrows_\eta$**.** The proof for $\leftrightarrows$ is extended to $\leftrightarrows_\eta$, maintaining its structure. We highlight the main differences.

We enrich $\mathcal{E}_{\mathcal{R}}$ with the equations corresponding to the two additional clauses of $\leftrightarrows_\eta$ (Definition 1.6). When $M \Downarrow x$ and $N \Downarrow \lambda z.\, N'$, where $N' \leftrightarrows_\eta xz$, we proceed as in case 4 of the definition of $\mathcal{E}_{\mathcal{R}}$, given that $N \leftrightarrows_\eta \lambda z.\, ((\lambda w.\, C_{\mathrm{e}}[w])(xV))$; the equation is:

$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\lambda z.\, \big((\lambda w.\, X_{w, C_{\mathrm{e}}[w]})\ (x\ X_{z,V})\big)]\!] \ .$$

We proceed likewise for the symmetric case.

In the optimised equations that we use to derive unique solutions, we add the following equation (relating values), as well as its symmetric counterpart:

$$\begin{array}{lcl} X^{\mathcal{V}}_{x,\lambda z.N'} & = & (y_0, \widetilde{y}) \\ & & !y_0(z,q).\, \overline{x}(z',q').\, (X^{\mathcal{V}}_{z,V}\langle z', \widetilde{y}'\rangle \mid q'(w).\, X_{w, C_{\mathrm{e}}[w]}\langle \widetilde{y}'', q\rangle) \ . \end{array}$$

Finally, to prove that $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ are solutions of $\mathcal{E}_{\mathcal{R}}$, we show that, whenever $M \Downarrow x$ and $N \Downarrow \lambda z.\, N'$, with $N' \Downarrow C_{\mathrm{e}}[xV]$:

$$\begin{array}{rl} \mathcal{I}[\![M]\!] & \approx E_{M,N}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]]\langle \widetilde{y}\rangle \\ & = \mathcal{I}[\![\lambda z.\, ((\lambda w.\, w)(xz))]\!] \end{array}$$

and

$$\begin{array}{rl} \mathcal{I}[\![N]\!] & \approx E_{M,N}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]]\langle \widetilde{y}\rangle \\ & = \mathcal{I}[\![\lambda z.\, ((\lambda w.\, C_{\mathrm{e}}[w])(xV))]\!] \ . \end{array}$$

To establish the former, we use algebraic reasoning to infer $\mathcal{I}[\![x]\!] \approx \mathcal{I}[\![\lambda z.\, xz]\!]$. For the latter, we use law (2) (given in the proof of Lemma 4.3).

Lemma 4.7 (Completeness for $\leftrightarrows_\eta$)**.** *For any M, N in Λ, $M \leftrightarrows_\eta N$ implies $\mathcal{I}[\![M]\!] \approx \mathcal{I}[\![N]\!]$.*

Combining Lemmas 4.2 and 4.7, and Theorem 1.11 we derive Full Abstraction for $\leftrightarrows_\eta$ with respect to barbed congruence.

Theorem 4.8 (Full Abstraction for $\leftrightarrows_\eta$)**.** *For any M, N in Λ, we have $M \leftrightarrows_\eta N$ iff $\mathcal{I}[\![M]\!] \simeq^{\mathrm{I}\pi} \mathcal{I}[\![N]\!]$*

Remark 4.9 (Unique solutions versus up-to techniques)**.** *For Milner's encoding of call-by-name λ-calculus, the completeness part of the full abstraction result with respect to Lévy-Longo Trees [26] relies on* up-to techniques for bisimilarity. *Precisely, given a relation $\mathcal{R}$ on λ-terms that represents a tree bisimulation, one shows that the π-calculus encoding of $\mathcal{R}$ is a π-calculus bisimulation* up-to context and expansion. *Expansion is a preorder that intuitively guarantees that a term is 'more efficient' than another one. In the up-to technique, expansion is used to manipulate the derivatives of two transitions so to bring up a common context. Such up-to technique is not powerful enough for the call-by-value encoding and the Eager Trees because some of the required transformations would violate expansion (i.e., they would require to replace a term by a 'less efficient' one). An example of this is law (2) (in the proof of Lemma 4.3), that would have to be applied from right to left so to implement the branching in clause (2) of Definition 1.3 (as a context with two holes).*

The use of the technique of unique solution of equations allows us to overcome the problem: law (2) and similar laws that introduce 'inefficiencies' can be used (and they are indeed used, in various places), as long as they do not produce new divergences.

5 Encoding into ALπ

Full abstraction with respect to η-Eager-Tree equality also holds for Milner's simplest encoding, namely $\mathcal{V}$ (Section 3), provided that the target language of the encoding is taken to be ALπ. The adoption of ALπ implicitly allows us to control capabilities, avoiding violations of laws such as (1) in the Introduction. In ALπ, bound output prefixes such as $\overline{a}(x).\, x(y)$ are abbreviations for $\boldsymbol{\nu} x\ (\overline{a}\langle x\rangle \mid x(y))$.

Theorem 5.1. *$M \leftrightarrows_\eta N$ iff $\mathcal{V}[\![M]\!] \simeq^{\mathrm{AL}\pi} \mathcal{V}[\![N]\!]$, for any $M, N \in \Lambda$.*

The main difference with respect to the proofs of Lemmas 4.6 and 4.7 is when proving absence of divergences for the (optimised) system of equations. Indeed, in ALπ the characterisation of barbed congruence ($\simeq^{\mathrm{AL}\pi}$) as bisimilarity makes use of a different labelled transition system where visible transitions may create new processes (the 'static links'), that could thus produce new reductions. Thus one has to show that the added processes do not introduce new divergences.

6 Contextual equivalence and preorders

We have presented full abstraction for η-Eager-Tree equality taking a 'branching' behavioural equivalence, namely barbed congruence, on the π-processes. We show here the same result for contextual equivalence, the most common 'linear' behavioural equivalence. We also extend the results to preorders.

We only discuss the encoding $\mathcal{I}$ into Iπ. Similar results however hold for the encoding $\mathcal{V}$ into ALπ.

6.1 Contextual relations and traces

Contextual equivalence is defined in the π-calculus analogously to its definition in the λ-calculus (Definition 1.2); thus, with respect to barbed congruence, the bisimulation game on reduction is dropped. Since we wish to handle preorders, we also introduce the *contextual preorder*.

Definition 6.1. Two Iπ agents A, B are in the *contextual preorder*, written $A \lesssim^{\mathrm{I}\pi}_{\mathrm{ct}} B$, if $C[A] \Downarrow_a$ implies $C[B] \Downarrow_a$, for all contexts C. They are *contextually equivalent*, written $A \simeq^{\mathrm{I}\pi}_{\mathrm{ct}} B$, if both $A \lesssim^{\mathrm{I}\pi}_{\mathrm{ct}} B$ and $B \lesssim^{\mathrm{I}\pi}_{\mathrm{ct}} A$ hold.

To manage contextual preorder and equivalence in proofs, we exploit characterisations of them as trace inclusion and equivalence. For $s = \mu_1, \ldots, \mu_n$, where each μ_i is a visible action, we set $P \overset{s}{\Longrightarrow}$ if $P \overset{\mu_1}{\Longrightarrow} P_1 \overset{\mu_2}{\Longrightarrow} P_2 \ldots P_{n-1} \overset{\mu_n}{\Longrightarrow} P_n$, for some processes $P_1, \ldots, P_n$.

Definition 6.2. Two Iπ processes P, Q are in the *trace inclusion*, written $P \preceq_{\mathrm{tr}} Q$, if $P \overset{s}{\Longrightarrow}$ implies $Q \overset{s}{\Longrightarrow}$, for each trace s. They are *trace equivalent*, written $P \approx_{\mathrm{tr}} Q$, if both $P \preceq_{\mathrm{tr}} Q$ and $Q \preceq_{\mathrm{tr}} P$ hold.

As usual, these relations are extended to abstractions by requiring instantiation of the parameters with fresh names.

Theorem 6.3. *In Iπ, relation $\lesssim^{\mathrm{I}\pi}_{\mathrm{ct}}$ coincides with $\preceq_{\mathrm{tr}}$, and relation $\simeq^{\mathrm{I}\pi}_{\mathrm{ct}}$ coincides with $\approx_{\mathrm{tr}}$.*

6.2 A proof technique for preorders

We modify the technique of unique solution of equations to reason about preorders, precisely the trace inclusion preorder.

In the case of equivalence, the technique of unique solutions exploits symmetry arguments, but symmetry does not hold for preorders. We overcome the problem by referring to the syntactic solution of the system in an asymmetric manner. This yields the two lemmas below, intuitively stating that the syntactic solution of a system is its smallest pre-fixed point, as well as, under the divergence-freeness hypothesis, its greatest post-fixed point. We say that $\widetilde{F}$ is a *pre-fixed point for* $\preceq_{\mathrm{tr}}$ of a system of equations $\{\widetilde{X} = \widetilde{E}\}$ if $\widetilde{E}[\widetilde{F}] \preceq_{\mathrm{tr}} \widetilde{F}$; similarly, $\widetilde{F}$ is a *post-fixed point for* $\preceq_{\mathrm{tr}}$ if $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{E}[\widetilde{F}]$.

Lemma 6.4 (Pre-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{\mathrm{K}}_{\mathcal{E}} \preceq_{\mathrm{tr}} \widetilde{F}$*

Lemma 6.5 (Post-fixed points, $\preceq_{\mathrm{tr}}$)**.** *Let $\mathcal{E}$ be a guarded system of equations, and $\widetilde{\mathrm{K}}_{\mathcal{E}}$ its syntactic solution. Suppose $\widetilde{\mathrm{K}}_{\mathcal{E}}$ has no divergences. If $\widetilde{F}$ is a post-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{\mathrm{K}}_{\mathcal{E}}$.*

Lemma 6.4 is immediate; the proof of Lemma 6.5 is similar to the proof of Theorem 2.4 (for bisimilarity). We thus derive the following proof technique.

Theorem 6.6. *Suppose that $\mathcal{E}$ is a guarded system of equations with a divergence-free syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, and $\widetilde{G}$ a post-fixed point, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{G}$.*

We can also extend Theorem 2.6 to preorders. We say that a system of equations $\mathcal{E}'$ *extends* $\mathcal{E}$ *with respect to a given preorder* if there exists a fixed set of indices J such that:

1. any pre-fixed point of $\mathcal{E}$ for the preorder can be obtained from a pre-fixed point of $\mathcal{E}'$ (for the same preorder) by removing the components corresponding to indices in J;
2. the same as (1) with post-fixed points in place of pre-fixed points.

Theorem 6.7. *Consider two systems of equations $\mathcal{E}'$ and $\mathcal{E}$ where $\mathcal{E}'$ extends $\mathcal{E}$ with respect to $\preceq_{\mathrm{tr}}$. Furthermore, suppose $\mathcal{E}'$ is guarded and has a divergence-free syntactic solution. If $\widetilde{F}$ is a pre-fixed point for $\preceq_{\mathrm{tr}}$ of $\mathcal{E}$, and $\widetilde{G}$ a post-fixed point, then $\widetilde{F} \preceq_{\mathrm{tr}} \widetilde{G}$.*

6.3 Full abstraction results

The preorder on λ-terms induced by the contextual preorder is *η-eager normal-form similarity*, $\preceq_\eta$. It is obtained by imposing that $M \preceq_\eta N$ for all N, whenever M is divergent. Thus, with respect to the bisimilarity relation $\leftrightarrows_\eta$, we only have to change clause (1) of Definition 1.3, by requiring only M to be divergent. (The bisimilarity $\leftrightarrows_\eta$ is then the intersection of $\preceq_\eta$ and its converse $\succeq_\eta$.)

Theorem 6.8 (Full abstraction on preorders)**.** *For any $M, N \in \Lambda$, we have $M \preceq_\eta N$ iff $\mathcal{I}[\![M]\!] \lesssim^{\mathrm{I}\pi}_{\mathrm{ct}} \mathcal{I}[\![N]\!]$.*

The structure of the proofs is similar to that for bisimilarity, using however Theorem 6.6. We discuss the main aspects of the completeness part.

Given an η-eager normal-form simulation $\mathcal{R}$, we define a system of equations $\mathcal{E}_{\mathcal{R}}$ as in Section 4.2. The only notable difference in the definition of the equations is in the case where $M\mathcal{R}N$, M diverges and N has an eager normal form. In this case, we use the following equation instead:

$$X_{M,N} = (\widetilde{y})\ \mathcal{I}[\![\Omega]\!] \quad . \tag{3}$$

As in Section 4.2, we define a system of guarded equations $\mathcal{E}'_{\mathcal{R}}$ whose syntactic solutions do not diverge. Equation (3) is replaced with $X_{M,N} = (\widetilde{y}, p)\ \mathbf{0}$.

Exploiting Theorem 6.7, we can use unique solution for preorders (Theorem 6.6) with $\mathcal{E}_{\mathcal{R}}$ instead of $\mathcal{E}'_{\mathcal{R}}$.

Defining $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ and $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$ as previously, we need to prove that $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!] \preceq_{\mathrm{tr}} \widetilde{E_{\mathcal{R}}}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]]$ and $\widetilde{E_{\mathcal{R}}}[\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]] \preceq_{\mathrm{tr}} \mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_2]\!]$. The former result is established along the lines of the analogous result in Section 4.2: indeed, $\mathcal{I}^{\mathrm{c}}[\![\mathcal{R}_1]\!]$ is a solution of $\mathcal{E}_{\mathcal{R}}$ for $\approx$, and $\approx_{\mathrm{tr}}$ is coarser than $\approx$.

For the latter, the only difference is due to equation (3), when $M\mathcal{R}N$, and M diverges but not N. In that case, we have to prove that $\mathcal{I}[\![\Omega]\!] \preceq_{\mathrm{tr}} \mathcal{I}[\![N]\!]$, which follows easily because the only trace of $\mathcal{I}[\![\Omega]\!]$ is the empty one, hence $\mathcal{I}[\![\Omega]\!]\langle p\rangle \preceq_{\mathrm{tr}} P$ for any P.

Corollary 6.9 (Full abstraction for $\simeq^{\mathrm{I}\pi}_{\mathrm{ct}}$)**.** *For any M, N in Λ, $M \leftrightarrows_\eta N$ iff $\mathcal{I}[\![M]\!] \simeq^{\mathrm{I}\pi}_{\mathrm{ct}} \mathcal{I}[\![N]\!]$.*

7 Conclusions and future work

In the paper we have studied the main question raised in Milner's landmark paper on functions as π-calculus processes, which is about the equivalence induced on λ-terms by their process encoding. We have focused on call-by-value, where the problem was still open; as behavioural equivalence on π-calculus we have taken contextual equivalence and barbed congruence (the most common 'linear' and 'branching' equivalences).

First we have shown that some expected equalities for open terms fail under Milner's encoding. We have considered two ways for overcoming this issue: rectifying the encodings (precisely, avoiding free outputs); restricting the target language to ALπ, so to control the capabilities of exported names. We have proved that, in both cases, the equivalence induced is Eager-Tree equality, modulo η (i.e., Lassen's η-eager normal-form bisimulation). We have then introduced a preorder on these trees, and derived similar full abstraction results for them with respect to the contextual preorder on π-terms. The paper is also a test case for the technique of unique solution of equations (and inequations), which is essential in all our completeness proofs.

Lassen had introduced Eager Trees as the call-by-value analogous of Lévy-Longo and Böhm Trees. The results in the paper confirm the claim, on process encodings of λ-terms: it was known that for (weak and strong) call-by-name, the equalities induced are those of Lévy-Longo Trees and Böhm Trees [29].

For controlling capabilities, we have used ALπ. Another possibility would have been to use a type system. In this case however, the technique of unique solution of equations needs to be extended to typed calculi. We leave this for future work.

We also leave for future work a thorough comparison between the technique of unique solution of equations and techniques based on enhancements of the bisimulation proof method (the "up-to" proof techniques), including if and how our completeness results can be derived using the latter techniques. (We recall that the "up-to" proof techniques are used in the completeness proofs with respect to Lévy-Longo Trees and Böhm Trees for the *call-by-name* encodings. We have discussed the problems with call-by-value in Remark 4.9.) In any case, even if other solutions existed, for this specific problem the unique solution technique appears to provide an elegant and natural framework to carry out the proofs.

For our encodings we have used the polyadic π-calculus; Milner's original paper [17] used the monadic calculus (the polyadic π-calculus makes the encoding easier to read; it had not been introduced at the time of [17]). We believe that polyadicity does not affect the results in the paper (the possibility of autoconcurrency breaks full abstraction of the encoding of the polyadic π-calculus into the monadic one, but autoconcurrency does not appear in the encoding of λ-terms).

In the call-by-value strategy we have followed, the function is reduced before the argument in an application. Our results can be adapted to the case in which the argument runs first, changing the definition of evaluation contexts. The parallel call-by-value, in which function and argument can run in parallel (considered in [18]), appears more delicate, as we cannot rely on the usual notion of evaluation context.

Interpretations of λ-calculi into π-calculi appear related to game semantics [5, 9, 10]. In particular, for untyped call-by-name they both allow us to derive Böhm Trees and Lévy-Longo Trees [11, 20]. To our knowledge, game semantics exist based on typed call-by-value, e.g., [2, 9], but not in the untyped case. In this respect, it would be interesting to see whether the relationship between π-calculus and Eager Trees studied in this paper could help to establish similar relationships in game semantics.

Acknowledgments

This work has been supported by the European Research Council (ERC) under the Horizon 2020 programme (CoVeCe, grant agreement No 678157); the ANR under the programmes "Investissements d'Avenir" (ANR-11-IDEX-0007), LABEX MILYON (ANR-10-LABX-0070), and Elica (ANR-14-CE25-0005); and the Université Franco-Italienne under the programme Vinci.

References

[1] Samson Abramsky. 1987. The Lazy λ-calculus. In *Research Topics in Functional Programming*, D. Turner (Ed.). Addison Wesley, 65–117.

[2] Samson Abramsky and Guy McCusker. 1997. Call-by-Value Games. In *Proceedings of, CSL '97, Annual Conference of the EACSL, Selected Papers*, Vol. 1414. Springer, 1–17.

[3] Beniamino Accattoli and Giulio Guerrieri. 2016. Open Call-by-Value. In *Proc. of APLAS 2016 (Lecture Notes in Computer Science)*, Vol. 10017. Springer Verlag, 206–226.

[4] H.P. Barendregt. 1984. *The lambda calculus: its syntax and semantics.* North-Holland.

[5] Martin Berger, Kohei Honda, and Nobuko Yoshida. 2001. Sequentiality and the pi-Calculus. In *Proceedings of TLCA (Lecture Notes in Computer Science)*, Vol. 2044. Springer, 29–45.

[6] Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. 2010. Termination in Impure Concurrent Languages. In *Proc. 21th Conf. on Concurrency Theory (Lecture Notes in Computer Science)*, Vol. 6269. Springer, 328–342.

[7] Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. 2017. Divergence and Unique Solution of Equations. In *Proceedings of CONCUR 2017 (LIPIcs)*, Vol. 85. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 11:1–11:16.

[8] J. Roger Hindley and Jonathan P. Seldin. 1986. *Introduction to Combinators and Lambda-Calculus.* Cambridge University Press.

[9] Kohei Honda and Nobuko Yoshida. 1999. Game-Theoretic Analysis of Call-by-Value Computation. *Theor. Comput. Sci.* 221, 1-2 (1999), 393–456.

[10] J. M. E. Hyland and C.-H. Luke Ong. 1995. Pi-Calculus, Dialogue Games and PCF. In *Proceedings of FPCA 1995.* ACM, 96–107.

[11] Andrew D. Ker, Hanno Nickau, and C.-H. Luke Ong. 2003. Adapting innocent game models for the Böhm treelambda-theory. *Theor. Comput. Sci.* 308, 1-3 (2003), 333–366.

[12] Søren B. Lassen. 2005. Eager Normal Form Bisimulation. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings.* IEEE Computer Society, 345–354.

[13] Søren B. Lassen and Paul Blain Levy. 2007. Typed Normal Form Bisimulation. In *Proc. of Computer Science Logic CSL 2007 (Lecture Notes in Computer Science)*, Vol. 4646. Springer, 283–297.

[14] Jean-Jacques Lévy. 1975. An algebraic interpretation of the lambda beta-calculus and a labeled lambda-calculus. In *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975 (Lecture Notes in Computer Science)*, Vol. 37. Springer, 147–165.

[15] Giuseppe Longo. 1983. Set-theoretical models of lambda-calculus: theories, expansions, isomorphisms. *Annals of Pure and Applied Logic* 24, 2 (1983), 153 – 188.

[16] Massimo Merro and Davide Sangiorgi. 2004. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science* 14, 5 (2004), 715–767.

[17] Robin Milner. 1990. *Functions as processes.* Research Report RR-1154. INRIA.

[18] Robin Milner. 1992. Functions as Processes. *Mathematical Structures in Computer Science* 2, 2 (1992), 119–141.

[19] Robin Milner. 1993. The polyadic π-calculus: a tutorial. In *Logic and algebra of specification.* NATO ASI Series (Series F: Computer & Systems Sciences), Vol. 94. Springer, 203–246.

[20] C.-H. Luke Ong and Pietro Di Gianantonio. 2004. Games characterizing Levy-Longo trees. *Theor. Comput. Sci.* 312, 1 (2004), 121–142.

[21] Gordon D. Plotkin. 1975. Call-by-Name, Call-by-Value and the lambda-Calculus. *Theor. Comput. Sci.* 1, 2 (1975), 125–159.

[22] Simona Ronchi Della Rocca and Luca Paolini. 2004. *The Parametric Lambda Calculus - A Metamodel for Computation.* Springer.

[23] Davide Sangiorgi. 1993. *Expressing mobility in process algebras : first-order and higher-order paradigms.* Ph.D. Dissertation. University of Edinburgh, UK.

[24] Davide Sangiorgi. 1993. An investigation into Functions as Processes. In *Proc. of MFPS'93 (Lecture Notes in Computer Science)*, Vol. 802. Springer, 143–159.

[25] Davide Sangiorgi. 1996. π-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theor. Comput. Sci.* 167, 1&2 (1996), 235–274.

[26] Davide Sangiorgi. 2000. Lazy functions and mobile processes. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner.* The MIT Press, 691–720.

[27] Davide Sangiorgi. 2006. Termination of processes. *Mathematical Structures in Computer Science* 16, 1 (2006), 1–39.

[28] Davide Sangiorgi and David Walker. 2001. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press.

[29] Davide Sangiorgi and Xian Xu. 2014. Trees from Functions as Processes. In *Proceedings of CONCUR 2014 (Lecture Notes in Computer Science)*, Vol. 8704. Springer, 78–92.

[30] Kristian Støvring and Søren B. Lassen. 2009. A Complete, Co-inductive Syntactic Theory of Sequential Control and State. In *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Vol. 5700. Springer, 329–375.

[31] Bernardo Toninho and Nobuko Yoshida. 2018. On Polymorphic Sessions and Functions - A Tale of Two (Fully Abstract) Encodings. In *Proc. of ESOP 2018 (Lecture Notes in Computer Science)*, Vol. 10801. Springer, 827–855.

[32] Nobuko Yoshida, Martin Berger, and Kohei Honda. 2004. Strong normalisation in the pi -calculus. *Inf. Comput.* 191, 2 (2004), 145–202.