Eager Functions as Processes
Adrien Durier, Daniel Hirschkoff, Davide Sangiorgi

To cite this version:

Adrien Durier, Daniel Hirschkoff, Davide Sangiorgi. Eager Functions as Processes. the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), Jul 2018, Oxford, United Kingdom. 10.1145/3209108.3209152. hal-01917255v2

HAL Id: hal-01917255
https://hal.science/hal-01917255v2
Submitted on 14 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 \( \lambda \)-calculus into the \( \pi \)-calculus. We show that, by tuning the encoding to two subcalculi of the \( \pi \)-calculus (Internal \( \pi \) and Asynchronous Local \( \pi \)), the equivalence on \( \lambda \)-terms induced by the encoding coincides with Lassen’s eager normal-form bisimilarity, extended to handle \( \eta \)-equality. As behavioural equivalence in the \( \pi \)-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 \( \lambda \)-calculus and call-by-value \( \lambda \)-calculus [1, 21] can be faithfully mimicked in the \( \pi \)-calculus, is generally considered a landmark in Concurrency Theory, and more generally in Programming Language Theory. The comparison with the \( \lambda \)-calculus is a significant expressiveness test for the \( \pi \)-calculus. More than that, it promotes the \( \pi \)-calculus to be a basis for general-purpose programming languages in which communication is the fundamental computing primitive. From the \( \lambda \)-calculus point of view, the comparison provides the means to study \( \lambda \)-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 \( \pi \)-calculus.

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

The main problem that Milner work left open is the characterisation of the equivalence on \( \lambda \)-terms induced by the encoding, whereby two \( \lambda \)-terms are equal if their encodings are behaviourally equivalent \( \pi \)-calculus terms. The question is largely independent of the precise form of behavioural equivalence adopted in the \( \pi \)-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 \( \lambda \)-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 \( \lambda \)-abstraction, and by including divergence among the observables [29]. These results show that, at least for call-by-name, the \( \pi \)-calculus encoding, while not fully abstract for the contextual equivalence of the \( \lambda \)-calculus, is in remarkable agreement with the theory of the \( \lambda \)-calculus: several well-known models of the \( \lambda \)-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 \( \lambda \) into \( \pi \) 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 \( \lambda \)-terms by their call-by-value encoding into the \( \pi \)-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 \( \eta \)-expansion, called \( \eta \)-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 \( \lambda \)-terms with free variables: desirable call-by-value equalities fail. An example is given by the law:

\[
I(xV) = xV
\]

where \( I = \lambda x. z \) and \( V \) is a value. Two possible solutions are:

1. rule out the free outputs; this essentially means transplanting the encoding onto the Internal \( \pi \)-calculus [25], a version of the \( \pi \)-calculus in which any name emitted in an output is fresh;

2. control the use of capabilities in the \( \pi \)-calculus; for instance taking Asynchronous Local \( \pi \) [16] as the target of the translation. (Controlling capabilities allows one to impose a directional dependency 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 \( \eta \)-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 \( \pi \)-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 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 a 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 \( \lambda \)-terms induced by a behavioural preorder on their \( \pi \)-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 \( \lambda \)-terms induced by the call-by-value encoding, when the behavioural relation on \( \pi \)-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 \( \lambda \)-calculus is contextual equivalence. Encodings into the \( \pi \)-calculus (be it for call-by-name or call-by-value) break contextual equivalence because \( \pi \)-calculus contexts are richer than those in the (pure) \( \lambda \)-calculus. In the paper we try to understand how far beyond contextual equivalence the discriminating power of the \( \pi \)-calculus brings us, for call-by-value. The opposite approach is to restrict the set of ‘legal’ \( \pi \)-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 \( \lambda \)-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 \( \lambda \)-calculus and the \( \pi \)-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 \( \pi \)-calculus: this is discussed 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 \( \pi \)-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 \( \mathcal{R} \circ \mathcal{R}' \). We use infix notation for relations; thus \( \mathcal{P} \mathcal{R} \mathcal{Q} \) means \( \mathcal{P} \circ \mathcal{R} \circ \mathcal{Q} \). A tilde represents a tuple. The \( i \)-th element of a tuple \( \bar{P} \) is referred to as \( P_i \). Our notations are extended to tuples componentwise. Thus \( \mathcal{P} \circ \mathcal{R} \circ \mathcal{Q} \) means \( P_1 \circ \mathcal{R} \circ Q_1 \) for all components.

1.1 The call-by-value \( \lambda \)-calculus

We let \( x \) and \( y \) range over the set of \( \lambda \)-calculus variables. The set \( \Lambda \) of \( \lambda \)-terms is defined by the grammar

\[
\begin{align*}
M &::= x \mid \lambda x. M \mid M_1 M_2 .
\end{align*}
\]

Free variables, closed terms, substitution, \( \alpha \)-conversion etc. are defined as usual [4, 8]. Here and in the rest of the paper (including when reasoning about \( \pi \) 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 \( fv(M) \). We group brackets on the left; therefore \( MNL \) is \((MN)L\). We abbreviate \( \lambda x_1 \ldots \lambda x_n. M \) as \( \lambda x_1 \ldots x_n. M \), or \( \lambda \bar{x}. M \) if the length of \( \bar{x} \) is not important. Symbol \( \Omega \) stands for the always-divergent term \( (\lambda x. x)(\lambda x. x) \).

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 \( \lambda \)-calculus values are abstractions and variables.

Evaluation contexts \( C_e ::= [\cdot] \mid C_e M \mid VC_e \)

Values \( V ::= x \mid \lambda x. M \)

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

Eager reduction (or \( \beta \)-reduction), \( \rightarrow \subseteq \Lambda \times \Lambda \), is determined by the rule:

\[
C_e[(\lambda x. M)V] \rightarrow C_e[M[V/x]] .
\]
We recall eager normal-form bisimilarity \( \Omega \) (Contextual equivalence)

**Definition 1.2** If \( M \) is a term in eager normal form, or there is an infinite reduction sequence \( (Eager normal-form bisimulation) \) starting from \( M \), then we say that \( M \) is contextually equivalent, written \( M \simeq_N \), if for any context \( C \), we have \( C[M] \Downarrow \) if \( C[N] \Downarrow \).

**Definition 1.3** (Eager normal-form bisimulation). A relation \( \simeq \) between \( \lambda \)-terms is an eager normal-form bisimulation if, whenever \( M \simeq N \), 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 \( V \vdash V' \) and \( C_e[z] \vdash C'_e[z] \) for a fresh \( z \);
3. \( M \Downarrow \lambda x. M' \) and \( N \Downarrow \lambda x. N' \) for some \( x, M', N' \) with \( M' \simeq N' \);
4. \( M \Downarrow x \) and \( N \Downarrow x \) for some \( x \).

**Eager normal-form bisimilarity**, \( \simeq \), is the largest eager normal-form bisimulation.

Essentially, the structure of a \( \lambda \)-term that is unveiled by Definition 1.3 is that of a (possibly infinite) tree obtained by repeatedly applying \( \beta \)-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 \( \simeq \) is strictly finer than contextual equivalence \( \simeq^{c1} \): the inclusion \( \simeq \subseteq \simeq^{c1} \) follows from the congruence properties of \( \simeq \) [12]; for the strictness, examples are the following equalities, that hold for \( \simeq^{c1} \) but not for \( \simeq \):

\[
\Omega = (\lambda y. \Omega)(xV) = (\lambda y. xV)(xV).
\]

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

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

**Definition 1.6** (\( \eta \)-eager normal-form bisimulation). A relation \( \simeq \) between \( \lambda \)-terms is an \( \eta \)-eager normal-form bisimulation if, whenever \( M \simeq N \), 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 \( y \vdash V \) and \( z \vdash C_e[z] \) 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 \( V \vdash y \) and \( C_e[z] \vdash z \) for some value \( V \), evaluation context \( C_e \), and fresh \( z \).

Then \( \eta \)-eager normal-form bisimilarity, \( \equiv \), is the largest \( \eta \)-eager normal-form bisimulation.

We sometimes call relation \( \equiv \) the \( \eta \)-Eager-Tree equality.

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

\[
x \equiv \eta \lambda y. xy \equiv \eta \lambda y. x(\lambda z. yz) \equiv \eta \lambda y. x(\lambda z. y(\lambda w. zw)) \equiv \eta \ldots
\]

A concrete example is given by taking a fixpoint \( Y \), and setting \( f \overset{\text{def}}{=} (\lambda xyz. x(zy)) \). We then have \( Y f x \equiv \lambda y. x(Y f y) \), and then \( x(Y f y) \equiv x(\lambda z. y f z) \), and so on. Hence, we have \( x \equiv \eta Y f x \).

### 1.3 The \( \pi \)-calculus, Iσ and AL\( \pi \)

In all encodings we consider, the encoding of a \( \lambda \)-term is parametric on a name, i.e., it is a function from names to \( \pi \)-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 \( F(\tilde{a}) \). We use \( P, Q \) for processes, \( F \) for abstractions. Processes and abstractions form the set of \( \pi \)-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 \( \pi \)-calculus is thus:

\[
\begin{align*}
A & \::= P \mid F \\
\text{(agents)} \\
P & \::= 0 \mid a(\tilde{b}) \mid \text{I}_\sigma(\tilde{b}) \mid P \mid \nu a P \\
& \mid P_1 \mid P_2 \mid !a(\tilde{b}) P \mid F(\tilde{a}) \\
\text{(processes)} \\
F & \::= (\tilde{a}) P \mid K \\
\text{(abstractions)}
\end{align*}
\]

In prefixes \( a(\tilde{b}) \) and \( \text{I}_\sigma(\tilde{b}) \), we call \( a \) the subject and \( \tilde{b} \) the object. When the tilde is empty, the surrounding brackets in prefixes will be omitted. We often abbreviate \( \nu a b P \) as \( (\nu a b) P \). An input prefix \( a(\tilde{b}) \), a restriction \( \nu b P \), and an abstraction \( (\tilde{b}) P \) are binders for names \( b \) and \( \tilde{b} \), 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 \( \alpha \)-conversion. An agent is name-closed if it does not contain free names. As in the \( \lambda \)-calculus, following the usual Barendregt convention we identify processes or actions which only differ on the choice of the bound names. The symbol \( \equiv \) will mean "syntactic identity modulo \( \alpha \)-conversion". Sometimes, we use \( \equiv \) 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 \( K \overset{\text{def}}{=} (\tilde{x}) P \), where \( \tilde{x} \) is name-closed; \( \tilde{x} \) 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
We will use two subcalculi: the Internal $\pi$-calculus ($I\pi$), and the Asynchronous Local $\pi$-calculus ($AL\pi$), obtained by placing certain constraints on prefixes.

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

$AL\pi$. $AL\pi$ is defined by enforcing that in an input $a(\overline{b})$, $P$, all names in $\overline{b}$ appear only in output position in $P$. Moreover, $AL\pi$ being asynchronous, output prefixes have no continuation; in the grammar of the $\pi$-calculus this corresponds to having only outputs of the form $\overline{a}(\overline{b})$, $0$ (which we will simply write $\overline{a}(\overline{b})$). 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 $I\pi$, on agents that are image-finite up to $\approx$, barbed congruence and bisimilarity coincide.

2. In $AL\pi$, 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 $\lambda$-calculus (into $I\pi$ and $AL\pi$) 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 $I\pi$ and $AL\pi$

We adapt the proof technique of unique solution of equations, from [7] to the calculi $I\pi$ and $AL\pi$, in order to derive bisimilarity results. The technique is discussed in [7] on the asynchronous $\pi$-calculus (for possibly-infinite systems of equations). The structure of the proofs for $I\pi$ and $AL\pi$ is similar; in particular the completeness part is essentially the same because bisimilarity is the same. The differences in the syntax of $I\pi$, and in the transition system of $AL\pi$, 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 $X(\overline{a})$). We use $E$ to range over such expressions; and $E$ 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 $X = E$) 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.

**Definition 1.10 (Bisimilarity).** A symmetric relation $R$ on $\pi$-processes is a bisimulation, if whenever $P \overset{\mu}{\to} P'$, then $P \overset{R}{\to} P'$ for some $Q'$.

Processes $P$ and $Q$ are bisimilar, written $P \approx Q$, if $P \overset{R}{\to} Q$ for some bisimulation $R$. 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 $\pi$ is a $\pi$-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 $C[A]$ is the agent resulting from replacing the holes with the terms $A$ or $A$.

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 $\pi$-calculus is standard [28] (including the labelled transition system). The reference behavioural equivalence for $\pi$-calculus will be the usual barbed congruence. We recall its definition, on a generic subset $L$ of $\pi$-calculus processes. A $L$-context is a process of $L$ 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 congruence).** Barbed bisimilarity is the largest symmetric relation $\approx$ on $\pi$-calculus processes such that $P \approx Q$ implies:

1. If $P \Rightarrow P'$ then there is $Q'$ such that $Q \Rightarrow Q'$ and $P' \approx Q'$.
2. $P \downarrow a$ iff $Q \downarrow a$.

Let $L$ be a set of $\pi$-calculus agents, and $A, B \in L$. We say that $A$ and $B$ are barbed congruent in $L$, written $A \approx L B$, if for each (well-sorted) $L$-context $C$, it holds that $C[A] \approx L C[B]$.

**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 $T$ of $\pi$-calculus processes which is derivation closed and such that $P \in T$ implies that, for all actions $\mu$, the set \{ $P' \mid P \overset{\mu}{\to} P'$ \} 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 $\lambda$-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 $R$ on $\pi$-processes is a bisimulation, if whenever $P \mathbin{R} Q$ and $P \overset{\mu}{\to} P'$, then $Q \overset{R}{\to} Q'$ for some $Q'$ with $P' \mathbin{R} Q'$.

Processes $P$ and $Q$ are bisimilar, written $P \approx Q$, if $P \mathbin{R} Q$ for some bisimulation $R$. 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. We adapt the proof technique of unique solution of equations, from [7] to the calculi $I\pi$ and $AL\pi$, in order to derive bisimilarity results. The technique is discussed in [7] on the asynchronous $\pi$-calculus (for possibly-infinite systems of equations). The structure of the proofs for $I\pi$ and $AL\pi$ is similar; in particular the completeness part is essentially the same because bisimilarity is the same. The differences in the syntax of $I\pi$, and in the transition system of $AL\pi$, 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 $X(\overline{a})$). We use $E$ to range over such expressions; and $E$ 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 $X = E$) 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[\tilde{F}] is the abstraction resulting from E by replacing each variable X_i with the abstraction \tilde{F}_i (as usual assuming \tilde{F} and \bar{X} have the same sort).

Definition 2.2. Suppose \{X_i = E_i\}_{i \in I} is a system of equations. We say that:
- \tilde{F} is a solution of the system of equations for \approx if for each i it holds that \tilde{F}_i \approx E_i[\tilde{F}].
- The system has a unique solution for \approx if whenever \tilde{F} and \tilde{G} are both solutions for \approx, we have \tilde{F} \approx \tilde{G}.

Definition 2.3 (Syntactic solutions). The syntactic solutions of the system of equations \bar{X} = \bar{E} are the recursively defined constants \tilde{K}_{E_i} \equiv E_i[\tilde{K}_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 \tau); 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} \ldots and for all i > n, \mu_i = \tau.

We call a divergence of P the sequence of transitions (P_i \xrightarrow{\mu} P_{i+1}),. In the case of an abstraction, \tilde{F} has a divergence if the process \tilde{F}(\tilde{a}) has a divergence, where \tilde{a} are fresh names.

Remark 2.7. We cannot derive Theorem 2.6 by comparing the syntactic solutions of the two systems \tilde{E}' and \tilde{E}. For instance, the equations X = \tau.X and X = \tau.X.\tau.\tau.\ldots 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 \lambda-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; \beta_\nu-reduction is then modeled as process interaction.

The encoding of a \lambda-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 \lambda-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{align*}
\mathcal{V}[\lambda x. M] & \overset{\text{def}}{=} (p) \tilde{F}(y).!(y(x), \mathcal{V}[M](q)) \\
\mathcal{V}[MN] & \overset{\text{def}}{=} (p)(wq)(\mathcal{V}[M](q) | q(y).\nu r (\mathcal{V}[N](r) | r(w).\tilde{F}(w, p))) \\
\mathcal{V}[x] & \overset{\text{def}}{=} (p) \tilde{F}(x)
\end{align*}

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

\begin{align*}
\mathcal{V}'[xy] & \overset{\text{def}}{=} (p) \tilde{F}(y).!(y(z), \tilde{F}(z), q).
\end{align*}

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 \beta_\nu-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 \lambda x is replaced by a bound output coupled with a link to x (the process \lambda y (z, q).\tilde{F}(z, q), receiving at y and re-emitting at x). Thus \beta_\nu-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 to the same value. The same argument holds for their \( \lambda \)-closures, \( \lambda x. xV \) and \( \lambda x. I(xV) \). We recall that \( \approx^\pi \) is barbed congruence in the \( \pi \)-calculus.

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

\[
V\left[I(xV)\right] \approx^\pi V\left[xV\right] \land V\left[I\left(xV\right)\right] \approx^\pi V\left[xV\right].
\]

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

\[
\begin{align*}
\forall \, \forall \langle p, q \rangle. & \, (p, q) \left[ \Pi(y), y \right], \forall \, (r) \left[ I\left[N\right]\right] \left( r \right) | r(w), \left[ \Pi\left( w', p' \right) \right] | \left( w' \right) & \left( \left( w \right) \right) |
\end{align*}
\]

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 \( \text{AL\pi} \), but not in \( \pi \).

This problem can be avoided by iterating the transformation that takes us from \( V \) to \( V' \) (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 \( \pi \); 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 \( \text{AL\pi} \), and we need not modify the initial encoding \( V \). This situation is analysed in Section 5.

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

\[
\lambda y. xy = x
\]

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

## 4 Encoding in the Internal \( \pi \)-calculus

### 4.1 Encoding and soundness

Figure 1 presents the encoding into \( I_\pi \), 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 \( a \) and \( b \) 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:

\[
p = \begin{cases} 
(p, q) \left[ \Pi(y), y \right], \forall \, (r) \left[ I\left[N\right]\right] \left( r \right) | r(w), \left[ \Pi\left( w', p' \right) \right] | \left( w' \right) & \left( \left( w \right) \right) |
\end{cases}
\]

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

The encoding validates \( \beta_\pi \)-reduction.

**Lemma 4.1** (Validity of \( \beta_\pi \)-reduction). For any \( M, N \in \Lambda \), \( M \rightarrow N \) implies \( I\left[M\right] \approx I\left[N\right] \).

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 \( \eta \)-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 \( \pi \)-terms imply the observables for \( \eta \)-eager normal-form bisimilarity in the encoded \( \lambda \)-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 \( \eta \)-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 \( \pi \)-terms, roughly allowing us to derive from the bisimilarity of two parallel compositions the componentwise bisimilarity of the single components. For the \( \eta \)-expansion, if \( I\left[\left[ x\right]\right] \approx I\left[\left[ \lambda z. M\right]\right] \), where \( M \equiv C_\eta xV \), we use a coinductive argument to derive \( V \equiv_\eta z \) and \( C_\eta \left[ y \right] \equiv y \), for \( y \) fresh; from this we then obtain \( \lambda z. M \equiv_\eta x \).

**Lemma 4.2** (Soundness). For any \( M, N \in \Lambda \), if \( I\left[M\right] \approx I\left[N\right] \) then \( M \equiv_\eta N \).

### 4.2 Completeness and Full Abstraction

To ease the reader into the proof, we first show the completeness for \( \approx_\pi \), rather than \( \approx_\eta \).

**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 \( \left( \overline{a} \right) \), then \( \left( \overline{a} \right) F \) is an abbreviation for its uncurrying \( \left( \overline{a} \right) F \).

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 \( \lambda \)-calculus to equation variables by setting

\[ I[[X_{M,N}]] \overset{\text{def}}{=} (p) X_{M,N}(\bar{y}, p) \quad \text{where } \bar{y} = \text{fv}(M,N). \]

We now describe the equation \( X_{M,N} = E_{M,N} \), for \((M,N) \in \mathcal{R}\). The equation is parameterised 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 \( \bar{y} = \text{fv}(M,N) \).

1. If \( M \parallel x \) and \( N \parallel x \), then the equation is the encoding of \(x\):

\[ X_{M,N} = (\bar{y}) I[[x]] = (\bar{y}, p) \mathcal{P}(z). z \triangleright x \]

2. If \( M \parallel \lambda x. M' \) and \( N \parallel \lambda x. N' \), then the equation uses a purely-divergent term; we choose the encoding of \(\Omega\):

\[ X_{M,N} = (\bar{y}) I[[\Omega]] \]

3. If \( M \parallel C_{c}[xV] \) and \( N \parallel \lambda x. N' \), then the equation encodes an abstraction whose body refers to the normal forms of \(M', N'\), via the variable \(X_{M',N'}\):

\[ X_{M,N} = (\bar{y}) I[[\lambda x. X_{M',N'}]] = (\bar{y}, p) \mathcal{P}(z). !(x,q). X_{M',N'}(\bar{y}', q) \]

4. If \( M \parallel C_{c}[xV] \) and \( N \parallel C'_{c}[xV'] \), 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 \(C_{c}[xV]\) into \((\lambda z. C_{c}[z])(xV')\), for some fresh \(z\), and similarly for \(C'_{c}\) and \(V'\); and (ii) referring to the variable for the evaluation contexts, \(X_{c_{c}[z], C_{c}[z]}\), and to the variable for the values, \(X_{V,V'}\). This yields the equation (for \(x\) fresh):

\[ X_{M,N} = (\bar{y}) I[[((\lambda z. C_{c}[z])(xV'))]] \quad (x \text{ fresh}) \]

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

1. \( X_{\lambda x. M} = (x') \quad I[[\lambda x. X_{x,M}]]\]

\[ = (x', p) \mathcal{P}(y). !(y,q). X_{x,M}(x', q) \]

2. \( X_{x,M} = (x, x') \quad I[[x]]\]

\[ = (x, x', p) \mathcal{P}(y). y \triangleright x \]

**Solutions of \( E_{\mathcal{R}}\).** 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 \(\mathcal{R}\) takes into account the variable 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 \(\bar{y}_j\) is fv\((M_j, N_j)\).

We write \(I^c[[\mathcal{R}_i]]\) for the closed abstractions resulting from the encoding of \(\mathcal{R}_i\), i.e., the tuple whose \(j\)-th component is \(\bar{y}_j\) \(I[[M_j]]\), and similarly for \(I^c[[\mathcal{R}_j]]\).

**Lemma 4.3.** \(I^c[[\mathcal{R}_1]]\) and \(I^c[[\mathcal{R}_2]]\) are solutions of \(E_{\mathcal{R}}\).

**Proof.** We show that each component of \(I^c[[\mathcal{R}_i]]\) is a solution of the corresponding equation, i.e., for the \(j\)-th component we show \((\bar{y}_j) \quad I[[M_i]] \approx E_{M_i,N_i} \quad I^c[[\mathcal{R}_i]]\).

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

\[ I[[((\lambda z. C_{c}[z])(xV'))]] \approx I[[C_{c}[xV]]], \quad (z \text{ fresh}) \quad (2) \]

We also exploit the validity of \(\beta_x\) for \(\approx\) (Lemma 4.1).

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

1. When \( M \parallel x \) and \( N \parallel x \), the equation is

\[ X_{M,N} = (\bar{y}, p) \quad 0 \]

2. When \( M \parallel V \) and \( N \parallel V' \), we introduce a new equation variable \(X_{V',V'}^V\) and a new equation; this will allow us, in the following step (3), to perform some optimisations. The equation is

\[ X_{M,N} = (\bar{y}, p) \mathcal{P}(z). X_{V',V'}^V(z, \bar{y}') \]

and we have, accordingly, the two following additional equations corresponding to the cases where values are functions or variables:

\[ X_{\lambda x. M, \lambda x. N'}^V = (z, \bar{y}) !z(x,q). X_{M', N'}(\bar{y}', q) \quad X_{x,x}^V = (z,x) \triangleright x \]

3. When \( M \parallel C_{c}[xV] \) and \( N \parallel C_{c}[xV'] \), we refer to \(X_{V',V'}^V\), instead of \(X_{V', V'}\), to remove all initial reductions in the corresponding equation for \(E_{\mathcal{R}}\). The first action thus becomes an output

\[ X_{M,N} = (\bar{y}, p) \mathcal{P}(z,q). (X_{V',V'}^V(z, \bar{y}) | q(w). X_{C_{c}[w], C_{c}[w]}(\bar{y}', p)) \]

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 \(E'_{\mathcal{R}}\) extends the system of equations \(E_{\mathcal{R}}\).

**Proof.** The new system \(E'_{\mathcal{R}}\) is obtained from \(E_{\mathcal{R}}\) 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.** \(E'_{\mathcal{R}}\) has a unique solution.

**Proof.** Divergence-freedom for the syntactic solutions of \(E'_{\mathcal{R}}\) 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 \(\tau\)-transition can ever be performed, after any number of visible actions. Further, \(E'_{\mathcal{R}}\) is guarded. Hence we can apply Theorem 2.4.

**Lemma 4.6** (Completeness for \(\approx\)). \(M \equiv N \text{ implies } I[[M]] \approx I[[N]], \text{ for any } M, N \in \Lambda.\)
We proceed likewise for the symmetric case.

Theorem 4.8

Expansion is used to manipulate the derivatives of two transitions so that the target language of the encoding is taken to be AL\(\pi\) (Section 3), provided that the target language of the encoding is taken to be AL\(\pi\). The adoption of AL\(\pi\) implicitly allows us to control capabilities, avoiding violations of laws such as (1) in the Introduction. In AL\(\pi\), bound output prefixes such as \(\pi(x).y\) are abbreviations for \(vx\ (\pi(x) | x(y))\).

Theorem 5.1. \(M \equiv_\eta N \iff \forall V[[M]] \equiv_{AL\pi} V[[N]],\) 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 AL\(\pi\) the characterisation of barbed congruence (\(\equiv_{AL\pi}\)) 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 \(\eta\)-Eager-Tree equality taking a ‘branching’ behavioural equivalence, namely barbed congruence, on the \(\pi\)-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 \(I\) into \(\pi\). Similar results however hold for the encoding \(V\) into AL\(\pi\).

6.1 Contextual relations and traces

Contextual equivalence is defined in the \(\pi\)-calculus analogously to its definition in the \(\lambda\)-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 \(\pi\) agents \(A, B\) are in the contextual preorder, written \(A \preceq_{ct} B\), if \(C[A] \not\parallel_{ct} B\) implies \(C[B] \not\parallel_{ct} A\), for all contexts \(C\). They are contextually equivalent, written \(A \equiv_{ct} B\), if both \(A \preceq_{ct} B\) and \(B \preceq_{ct} A\) 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 \(\mu_i\) is a visible action, we set \(P \overset{s}{\Rightarrow} P_1 \overset{\mu_1}{\Rightarrow} P_2 \overset{\mu_2}{\Rightarrow} \cdots \overset{\mu_{n-1}}{\Rightarrow} P_n \Rightarrow P_n\), for some processes \(P_1, \ldots, P_n\).

Definition 6.2. Two \(\pi\) processes \(P, Q\) are in the trace inclusion, written \(P \preceq_{tr} Q\), if \(P \Rightarrow s\) implies \(Q \Rightarrow s\), for each trace \(s\). They are trace equivalent, written \(P \equiv_{tr} Q\), if both \(P \preceq_{tr} Q\) and \(Q \preceq_{tr} P\) hold.

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

Theorem 6.3. In \(\pi\), relation \(\preceq_{ct}\) coincides with \(\preceq_{tr}\), and relation \(\equiv_{ct}\) coincides with \(\equiv_{tr}\).
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 \( \tilde{F} \) is a pre-fixed point for \( \preceq_{\text{tr}} \) of a system of equations \( \{ \tilde{X} = \tilde{E} \} \) if \( \tilde{E}(\tilde{F}) \preceq_{\text{tr}} \tilde{F} \); similarly, \( \tilde{F} \) is a post-fixed point for \( \preceq_{\text{tr}} \) if \( \tilde{F} \preceq_{\text{tr}} \tilde{E}(\tilde{F}) \).

Lemma 6.4 (Pre-fixed points, \( \preceq_{\text{tr}} \)). Let \( E \) be a system of equations, and \( K_E \), its syntactic solution. If \( \tilde{F} \) is a pre-fixed point for \( \preceq_{\text{tr}} \) of \( E \), then \( K_E \preceq_{\text{tr}} \tilde{F} \).

Lemma 6.5 (Post-fixed points, \( \preceq_{\text{tr}} \)). Let \( E \) be a guarded system of equations, and \( K_E \), its syntactic solution. Suppose \( K_E \) has no divergences. If \( \tilde{F} \) is a post-fixed point for \( \preceq_{\text{tr}} \) of \( E \), then \( \tilde{F} \preceq_{\text{tr}} K_E \).

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 \( E \) is a guarded system of equations with a divergence-free syntactic solution. If \( \tilde{F} \) is a pre-fixed point for \( \preceq_{\text{tr}} \) of \( E \), and \( \tilde{G} \) a post-fixed point, then \( \tilde{F} \preceq_{\text{tr}} \tilde{G} \).

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

1. any pre-fixed point of \( E \) for the preorder can be obtained from a pre-fixed point of \( E' \) (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 \( E' \) and \( E \) where \( E' \) extends \( E \) with respect to \( \preceq_{\text{tr}} \). Furthermore, suppose \( E' \) is guarded and has a divergence-free syntactic solution. If \( \tilde{F} \) is a pre-fixed point for \( \preceq_{\text{tr}} \) of \( E \), and \( \tilde{G} \) a post-fixed point, then \( \tilde{F} \preceq_{\text{tr}} \tilde{G} \).

6.3 Full abstraction results
The preorder on \( \lambda \)-terms induced by the contextual preorder is \( \eta \)-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 \( \equiv_\eta \), we only have to change clause (1) of Definition 1.3, by requiring only \( M \) to be divergent. (The bisimilarity \( \equiv_\eta \) is then the intersection of \( \preceq_\eta \) and its converse \( \preceq_\eta ^{-1} \).)

Theorem 6.8 (Full abstraction on preorders). For any \( M, N \in \Lambda \), we have \( M \preceq_{\text{tr}} N \iff I([M]) \preceq_{\text{tr}} I([N]) \).

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 \( \eta \)-eager normal-form simulation \( R \), we define a system of equations \( E_R \) as in Section 4.2. The only notable difference in the definition of the equations is in the case where \( M R N \), \( M \) diverges and \( N \) has an eager normal form. In this case, we use the following equation instead:

\[
X_{M,N} = (\bar{g} I [\Omega]) .
\]

As in Section 4.2, we define a system of guarded equations \( E_R' \) whose syntactic solutions do not diverge. Equation (3) is replaced with \( X_{M,N} = (\bar{g} p) 0 \).

Exploiting Theorem 6.7, we can use unique solution for preorders (Theorem 6.6) with \( E_R \) instead of \( E_R' \).

Defining \( I^E([R_1]) \) and \( I^E([R_2]) \) as previously, we need to prove that \( I^E([R_1]) \preceq_{\text{tr}} E_R(I^E([R_1])) \) and \( E_R(I^E([R_2])) \preceq_{\text{tr}} I^E([R_2]) \). The former result is established along the lines of the analogous result in Section 4.2: indeed, \( I^E([R_1]) \) is a solution of \( E_R \) for \( \preceq \), and \( \preceq_{\text{tr}} \) is coarser than \( \preceq \).

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

Corollary 6.9 (Full abstraction for \( \preceq_{\text{ct}} \)). For any \( M, N \in \Lambda, M \equiv_\eta N \iff I([M]) \preceq_{\text{ct}} I([N]) \).

7 Conclusions and future work
In the paper we have studied the main question raised in Milner’s landmark paper on functions as \( \pi \)-calculus processes, which is about the equivalence induced on \( \lambda \)-terms by their process encoding. We have focused on call-by-value, where the problem was still open; asBehavioural equivalence on \( \pi \)-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 AL, so to control the capabilities of exported names. We have proved that, in both cases, the equivalence induced by Eager-Tree equality, modulu \( \eta \) (i.e., Lassen’s \( \eta \)-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 \( \pi \)-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 \( \lambda \)-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 AL. 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, 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.

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