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

To cite this version:
Adrien Durier, Daniel Hirschkoff, Davide Sangiorgi. Eager Functions as Processes. 2018. <hal-01736696>

HAL Id: hal-01736696
https://hal.archives-ouvertes.fr/hal-01736696
Submitted on 18 Mar 2018

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

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

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 equival-
ence on λ-terms induced by the encoding coincides with Lassen’s
eager normal-form bisimilarity, extended to handle η-equivalence. 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-intro-
duced 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 [16, 17], that shows how
the evaluation strategies of call-by-name λ-calculus and call-by-
value λ-calculus [1, 20] 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 instru-
mments 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 equiv-
alent, 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 character-
isation 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 conflu-
ent). In the paper we consider contextual equivalence (that coincides
with may testing and trace equivalence) and barbed congruence
(that coincides with bisimilarity).

LICS’18, July 9–12, 2018, Oxford
2018.

For the call-by-name λ-calculus, the answer was found shortly
later [24]: the equality induced is the equality of Levy-Longo Trees [14],
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 diver-
gence among the observables [26]. 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 Levy-Longo Trees or Böhm Trees as
their induced equivalence [4, 13, 14].

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 com-
pleteness 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 [11, 12]. This is a tree structure for call-by-value,
proposed by Lassen as the call-by-value counterpart of Levy-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:

\[ I(xV) = xV \]  \hspace{1cm} (1)

where \( I \) is λz. 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 [23], 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 [15] as the target of the trans-
lation. (Controlling capabilities allows one to impose a di-
rectionality on names, which, under certain technical condi-
tions, 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.

A key role in the completeness proof is played by the technique of
unique solution of equations, recently proposed [6]. This 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 pre-
order 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, 28].

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 [21].

In [6], the unique-solution technique is used in the completeness
proof for Milner’s call-by-name encoding. That proof essentially
revisits the proof of [24], 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, R ranges over relations. The composition of
two relations R and R’ is written RR’. We often use infix notation
for relations; thus P R Q means (P, Q) ∈ R. A tilde represents
a tuple. The i-th element of a tuple P is referred to as P_i. Our
notations are extended to tuples componentwise. Thus P Q means
P_i Q_j for all components. We anticipate that Appendix A
presents a summary of the behavioural relations used in this paper.

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

\[ M ::= x | \lambda x. M | M_1 M_2. \]

Free variables, closed terms, substitution, α-conversion etc. are de-
clined as usual [4, 7]. 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 fv(M). We group brackets on the left; therefore
MLN is (ML)N. We abbreviate \(x_1, \ldots, x_n\) M as \(x_1 \cdots x_n M\), or
\(\lambda x. M\) if the length of \(x\) 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 [], possibly occurring more than
once. If C is a context, C[M] is a shorthand for C where the hole
[] is substituted by M. An evaluation context is a special kind of
context, with exactly one hole [], and in which the inserted term
can immediately run. In the pure λ-calculus values are abstractions
and variables.

Evaluation contexts \[ C_e ::= [] | C_e M V C_e \]
Values \[ V ::= x | \lambda x. M \]

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

Eager reduction (or β-reduction), \(\to\) ⊆ Λ × Λ, is determined
by the rule:

\[ C_e[(\lambda x. M)V] \to C_e[M[V/x]] . \]

We write \(\to\) for the reflexive transitive closure of \(\to\). A term
in eager normal form is a term that has no eager reduction.

Proposition 1.1. 1. If \(M \to M’\), then \(C_e[M] \to C_e[M’]\)
and \(M_\sigma \to M’_\sigma\), for any value substitution \(\sigma\).
2. Terms in eager normal form are either values or of the shape
\(C_e[xV]\).

Therefore, given a term M, either M \(\to\) 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 \(\nmid\). We write M \(\Downarrow\) \(\mid\) when M \(\Downarrow\) M’ for some M’.

Definition 1.2 (Contextual equivalence). Given M, N ∈ Λ, we
say that M and N are contextually equivalent, written M \(\equiv^C\) 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 [11, 12, 27].

Definition 1.3 (Eager normal-form bisimulation). A relation R
between λ-terms is an eager normal-form bisimulation if, whenever
M R N, one of the following holds:
1. both M and N diverge;
Eager Functions as Processes

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 \( V \not\equiv V' \) and \( C_e(z) \not\equiv C'_e(z) \) for a fresh \( z \);

3. \( M \Downarrow \lambda x. M' \) and \( N \Downarrow \lambda x. N' \) for some \( x \), \( M', N' \) with \( M' \not\equiv \lambda x. N' \);

4. \( M \Downarrow x \) and \( N \Downarrow x \) for some \( x \).

**Eager normal-form bisimilarity, \( \equiv_e \),** 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 \( \equiv_e \) is strictly finer than contextual equivalence \( \equiv_c^\Delta \); the inclusion \( \equiv_e \subseteq \equiv_c^\Delta \) follows from the congruence properties of \( \equiv_e \) [11]; for the strictness, examples are the following equalities, that hold for \( \equiv_c^\Delta \) but not for \( \equiv_e \):

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

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

The failure of the \( \eta \)-rule \( \lambda y. x y \not\equiv x \) 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 [11] takes \( \eta \)-expansion into account so to recover such missing equalities.

**Definition 1.6 (\( \eta \)-eager normal-form bisimulation).** A relation \( R \) between \( \lambda \)-terms is an \( \eta \)-eager normal-form bisimulation if, whenever \( R \not\equiv \lambda \), either one of the clauses of Definition 1.3, or one of the two following additional clauses, hold:

5. \( M \Downarrow x \) and \( N \Downarrow x \) for some \( x \), \( y \), and \( N' \) such that \( N' \not\equiv C_e(xV) \), with \( y \not\equiv V \) and \( z \not\equiv C_e(z) \) for some value \( V \), evaluation context \( C_e \), and fresh \( z \).

6. the converse of (5), i.e., \( N \Downarrow x \) and \( M \Downarrow x \) for some \( x \), \( y \), and \( M' \) such that \( M' \not\equiv C_e(xV) \), with \( V \not\equiv y \) and \( C_e(z) \not\equiv z \) for some value \( V \), evaluation context \( C_e \), and fresh \( z \).

Then \( \eta \)-eager normal-form bisimilarity, \( \equiv_{\eta e} \), is the largest \( \eta \)-eager normal-form bisimulation.

We sometimes call relation \( \equiv_{\eta} \) 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,

\[
x \equiv_{\eta} \lambda y. x y \equiv_{\eta} \lambda y. x(\lambda z. y) \equiv_{\eta} \lambda y. x(\lambda z. y(\lambda w. zw)) \equiv_{\eta} \ldots
\]

A concrete example is given by taking a fixpoint \( Y \), and setting

\[
f \overset{\text{def}}{=} (\lambda x y. x(yy)) \quad \text{We then have } Y f x \equiv \lambda y. x(Y f y) \text{, and then } x(Y f y) \equiv x(\lambda z. y(Y f z)) \text{, and so on. Hence, we have } x \equiv_{\eta} Y f x.
\]

**1.3 The \( \pi \)-calculus, \( \eta \pi \) and \( \Lambda \pi \).**

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 \( F(a) \). We use \( P, Q \) for processes, \( F \) for abstractions. Processes and abstractions form the set of \( \pi \)-agents (or simply agents), ranged over by \( \Gamma \). Small letters \( a, b, \ldots, x, y, \ldots \) range over the infinite set of names. The grammar of the \( \pi \)-calculus is thus:

\[
A \overset{\text{def}}{=} P \mid F \quad (\text{agents})
\]

\[
P \overset{\text{def}}{=} 0 \mid a(b), P \mid \bar{a}(b), P \mid va P \quad (\text{processes})
\]

\[
F \overset{\text{def}}{=} (\bar{a}) P \mid K \quad (\text{abstractions})
\]

In prefixes \( a(b) \) and \( \bar{a}(b) \), we call \( a \) the subject and \( b \) the object. When the tilde is empty, the surrounding brackets in prefixes will be omitted. We often abbreviate \( va vb P \) as \( (va, b)P \). An input prefix \( a(b)P \), a restriction \( vb P \), and an abstraction \( (b)P \) are binders for names \( b \) 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 \( \tau \)-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 \( = \) will mean “syntactic identity modulo \( \alpha \)-conversion”. Sometimes, we use \( \equiv_{\def} \) 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 \( K \overset{\text{def}}{=} (x) P \), where \( x \) is name-closed; \( x \) 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 [18] 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 \( \pi \) is a \( \pi \)-agent in which some subterms have been replaced by the hole \([\ldots]\) or, if the context is polyadic, with indexed holes \([\ldots];_{1}, \ldots,_{n};_{1};_{2};_{n};_{1};_{2}\); then \( C | A \) or \( C | \bar{A} \) is the agent resulting from replacing the holes with the terms \( A \) or \( \bar{A} \).

We omit the operators of sum and matching (not needed in the encodings). We refer to [18] 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 [25] (including the labelled transition system), and given in Appendix B. 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 \([\ldots]\) in it (the hole has a sort too, as it could be in place of an abstraction). We write \( P \Downarrow_{a} P \) 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 asynchronos 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
\[
P \approx Q \text{ implies:}
\]
1. If \( P \Rightarrow P' \) then there is \( Q' \) such that \( Q \Rightarrow Q' \) and \( P' \approx Q' \).
2. \( P \parallel_a Q \Rightarrow P \parallel_a Q \).

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

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 \( \mathcal{F}[\pi] \) of \( \pi \)-calculus processes which is derivation closed and such that \( P \in \mathcal{F}[\pi] \) implies that, for all actions \( \mu \), the set \( \{ P' \mid P \red \mu P' \} \) 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 \( \mathcal{R} \) on \( \pi \)-processes is a bisimulation, if whenever \( P \mathcal{R} Q \) and \( P \red \mu P' \), then \( Q \red \mu Q' \) for some \( Q' \) with \( P' \mathcal{R} Q' \).

Processes \( P \) and \( Q \) are bisimilar, written \( P \equiv \mathcal{R} Q \), if \( P \mathcal{R} Q \) for some bisimulation \( \mathcal{R} \).

We will use two subcalculi: the internal \( \pi \)-calculus (\( \pi \)), and the Asynchronous Local \( \pi \)-calculus (AL\( \pi \)), obtained by placing certain constraints on prefixes.

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

AL\( \pi \). AL\( \pi \) is defined by enforcing that in an input \( a(\vec{b}) \), \( P \), all names in \( \vec{b} \) 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 \( \vec{a}(b) \). 0 (which we will simply write \( \vec{a}(b) \)). In AL\( \pi \), to maintain the characterisation of barbed congruence as (ground) bisimilarity, the transition system has to be modified \([15]\), allowing the dynamic introduction of additional processes (the ‘links’, sometimes also called forwarders). Details are given in Appendix C.

Theorem 1.11. In \( \pi \), 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 \( \pi \) 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 \( \pi \) and AL\( \pi \)

We adapt the proof technique of unique solution of equations, from \([6]\) to the calculi \( \pi \) and AL\( \pi \), in order to derive bisimilarity results. The technique is discussed in \([6]\) on the asynchronous \( \pi \)-calculus. The structure of the proofs for \( \pi \) 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 \( \pi \) 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 \( X(\bar{a}) \)). We use \( E \) to range over such expressions; and \( \mathcal{E} \) to range over systems of equations, defined as follows.

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 \( X = \bar{E} \)) 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 underneat a prefix.

\( E(\bar{F}) \) is the abstraction resulting from \( E \) by replacing each variable \( X_i \) with the abstraction \( F_i \) (as usual assuming \( \bar{F} \) 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:

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

Definition 2.3 (Syntactic solutions). The syntactic solutions of the system of equations \( X = \bar{E} \) are the recursively defined constants \( K_{E_i} \equiv E_i[\bar{K}_E] \), for each \( i \leq S \), where \( S \) is the size 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 \( r \)); formally, there are processes \( P_i, i \geq 0 \), and some \( n \), such that \( P = P_0 \red P_1 \red P_2 \red \ldots \) and for all \( i > n, \mu_i = r \). We call a divergence of \( P \) the sequence of transitions \( \{ P_i \red P_{i+1} \} \).

In the case of an abstraction, \( F \) has a divergence if the process \( F(\bar{a}) \) has a divergence, where \( \bar{a} \) are fresh names. A tuple of agents \( \bar{A} \) 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 λν and ALπ.

**Theorem 2.4.** In λν and ALπ, a guarded system of equations with divergence-free syntactic solutions has unique solution for \( \ast \).

### 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 \( E' \) extends system \( E \) if there exists a fixed set of indices \( J \) such that any solution of \( E \) can be obtained from a solution of \( E' \) by removing the components corresponding to indices in \( J \).

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

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 \( E' \) and \( E \). For instance, the equations \( X = \pi X \) and \( X = \pi \pi \ldots \) 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 [16, 17] 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; \( \beta_v \)-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, 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 [16], Milner presented two candidates for the encoding of call-by-value \( \lambda \)-calculus [20]. They follow the same idea of translation, but with a technical difference in the rule for variables. One encoding, \( V' \), is so defined:

\[
V'[\lambda x. M] \equiv (p) \overline{\lambda x.(y(q(x,q)).V'[M](q))}
\]

\[
V'[MN] \equiv (p)(\nu q)(V'[M](q) \mid \nu r (V'[N](r) \mid r(w).\overline{y}(w,p)))
\]

\[
V'[x] \equiv (p)\overline{x}(p)
\]

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

\[
V'[x] \equiv (p)\overline{x}(p).
\]

The encoding \( V' \) is more efficient than \( V' \), as it uses fewer communications.

#### 3.2 Some problems with open terms

The immediate free output in the encoding of variables in \( V \) breaks the validity of \( \beta_v \)-reduction; i.e., there exist a term \( M \) and a value \( V \) such that \( V'[\lambda x. M]V \neq V'[M[V/x]] \). The encoding \( V' \) 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 \( !y(q,z.q).\overline{x}(z,q) \), receiving at \( y \) and re-emitting at \( x \)). Thus \( \beta_v \)-reduction is validated [22]. (The final version of Milner’s paper [17], was written after the results in [22] were known and presents only the encoding \( V' \).)

Nevertheless, \( V' \) 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. We recall that \( \simeq \) is barbed congruence in the \( \pi \)-calculus.

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

\[
V'[I(xV)] \neq \simeq V'[xV] \land V'[I(xV)] \neq \simeq V'[xV']
\]

(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:

\[
V'[\overline{x}(V)](p) \simeq \overline{x}(V).\forall w (\overline{V}(w,p) \mid !w(u).\overline{y}(u))
\]

\[
V'[I(xy)](p) \simeq \overline{x}(V).\forall w,q(\overline{V}(w,q) \mid !w(u).\overline{y}(u) \mid q(z).\overline{z}(w').\overline{z}(w'))
\]

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 \( \pi \).

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 ALπ, 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 η-expansion — (the law fails for Milner’s encoding into the π-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 π-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 \( a \rightarrow b \) represents a link (sometimes called forwarder; for readability we have adopted the infix notation \( a \rightarrow b \) for the constant \( \rightarrow \)). It transforms all outputs at \( a \) into outputs at \( b \) (therefore \( a \) and \( b \) are names of the same sort). Thus the body of \( a \rightarrow b \) 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 \( \rightarrow \) therefore is:

\[
\begin{align*}
\rightarrow & \triangleq \begin{cases}
(p, q) p(x), \eta y(q). y \rightarrow x & \text{if \( p, q \) are continuation names} \\
(x, y) \lambda x(p, z), \eta y(q, w). (q \rightarrow p | w \rightarrow z) & \text{otherwise}
\end{cases}
\end{align*}
\]

(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 \( M, N \in \Lambda \), \( M \longrightarrow N \) implies \( I \llbracket M \rrbracket \approx I \llbracket N \rrbracket \).

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 [24]. 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 \( I \llbracket M \rrbracket \approx I \llbracket \lambda z. M \rrbracket \), where \( M \not\in \mathbb{C}_e \) and \( \mathbb{C}_e \) we use a coinductive argument to derive \( V \Rightarrow \eta \ z \) and \( C_\varepsilon[ \eta \ z ] \Rightarrow \eta \ y \), for \( y \) fresh; from this we then obtain \( \lambda z. M \Rightarrow \eta \ x \). More details for the proof of soundness are given in Appendix D.

**Lemma 4.2 (Soundness).** For any \( M, N \in \Lambda \), if \( I \llbracket M \rrbracket \approx I \llbracket N \rrbracket \) then \( M \Rightarrow \eta \ N \).

### 4.2 Completeness and Full Abstraction

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

**The system of equations.** Suppose \( \mathcal{R} \) is an eager normal-form bisimulation. We define a system of equations \( \mathcal{E}_R \), 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 \( \mathcal{E}_R \) 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 \( \Gamma \) is an abstraction, say \( \Gamma p \), then \( \Gamma y \) is an abbreviation for its uncurrying \( \Gamma y \ Gamma \).

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

\[
I \llbracket X_{M,N} \rrbracket \triangleq (p, \mu y). X_{M,N}(\bar{y},p) \quad \text{where } \bar{y} = \nu y \Gamma y \ y = \nu y M \Gamma y .
\]

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 \( \bar{y} = \nu y M \Gamma y \).

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

\[
X_{M,N} = (\bar{y}) I \llbracket x \rrbracket = (\bar{y}, p) \Gamma z. z \rightarrow x
\]

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

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

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 \( X_{M',N'} \),

\[
X_{M,N} = (\bar{y}) I \llbracket \lambda x. M', N' \rrbracket = (\bar{y}, p) \Gamma z. \Gamma x(z, q). X_{M',N'}(\bar{y}', q)
\]

4. If \( M \not\in \mathbb{C}_e \) and \( N \not\in \mathbb{C}_e \), 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 \( C_\varepsilon[\bar{y} \ v'] \) into \( (\lambda z. C_\varepsilon[z])(\bar{y} \ v') \), for some fresh \( z \) and similarly for \( C_\varepsilon' \) and \( v' \) (such a transformation is valid for \( \Rightarrow \)); and (ii) referring to the variable for the evaluation contexts, \( X_{C_\varepsilon[z]} C_\varepsilon[z] \) and to the variable for the values, \( X_{v', v'} \). This yields the equation
We also exploit the validity of $H$ when $X \mapsto x$, instead of $X',v'$, so to remove all initial reductions in the corresponding equation for $E_R$. The first action thus becomes an output:

$$X_{M,N} = (g, p) \exists (z, q). (X'_{v,v'}'(z, \tilde{y}) \mid q(w), X_{C[v], C[v]}(\tilde{y}'', p))$$

Lemmas 4.4 and 4.5 are obtained by applying Theorem 2.6. (In Lemma 4.4, ‘extend’ is as by Definition 2.5.)

**Lemma 4.4.** The system of equations $E'_R$ extends the system of equations $E_R$.

**Proof.** The new system $E'_R$ is obtained from $E_R$ 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.** $E'_R$ has a unique solution.

**Proof.** Divergence-freedom for the syntactic solutions of $E'_R$ 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, $E'_R$ is guarded. Hence we can apply Theorem 2.4. □

**Lemma 4.6** (Completeness for $\Rightarrow$). $M \Rightarrow N$ implies $I[M] \Rightarrow I[N]$, for any $M, N \in \Lambda$.

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

**Completeness for $\Rightarrow_{\eta}$.** The proof for $\Rightarrow$ is extended to $\Rightarrow_{\eta}$, maintaining its structure. We highlight the main differences.

We enrich $E_R$ with the equations corresponding to the two additional clauses of $\Rightarrow_{\eta}$ (Definition 1.6). When $M \Downarrow x$ and $N \Downarrow \lambda \cdot N'$, with $N' \Rightarrow_{\eta} x$, we proceed as in case 4 of the definition of $E_R$, given that $N \Rightarrow_{\eta} \lambda \cdot (\tilde{w}, X_{C[v]}(\tilde{v}))$; the equation is:

$$X_{M,N} = (g, p) \exists (z, q). (X'_{v,v'}'(z, \tilde{y}) \mid q(w), X_{C[v], C[v]}(\tilde{y}'', p))$$

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:

$$X'_{z,v,v'} \mid (g, p) \exists (z, q). X'_{z,v,v'}'(z, \tilde{y}) + (\tilde{w}, X_{C[v]}(\tilde{v}))\right)$$

Finally, to prove that $I[\mathcal{R}_1]$ and $I[\mathcal{R}_2]$ are solutions of $E_R$, we show that, whenever $M \Downarrow x$ and $N \Downarrow \lambda \cdot N'$, with $N' \Downarrow \lambda \cdot C[v]$:

$$I[M] \approx E_{M,N}[I[\mathcal{R}_1](\tilde{y})]$$

$$I[N] \approx E_{M,N}[I[\mathcal{R}_2](\tilde{y})]$$

and

$$I[M] \Rightarrow I[N]$$

The proof for $\Rightarrow_{\eta}$ is extended to $\Rightarrow_{\eta}$, maintaining its structure. We highlight the main differences.
To establish the former, we use algebraic reasoning to infer $I\left[ x \right] \cong I\left[ xz, xz \right]$. For the latter, we use law (2) (given in the proof of Lemma 4.3). More details are provided in Appendix D.3.

**Lemma 4.7** (Completeness for $\equiv_{\eta}$). For any $M, N$ in $\Lambda$, $M \equiv_{\eta} N$ implies $I\left[ M \right] \cong I\left[ N \right]$.

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

**Theorem 4.8** (Full Abstraction for $\equiv_{\eta}$). For any $M, N$ in $\Lambda$, we have $M \equiv_{\eta} N$ if $I\left[ M \right] \cong I\left[ N \right]$.

**Remark 4.9** (Unique solutions versus up-to techniques). For Milner’s encoding of call-by-name $\lambda$-calculus, the completeness part of the full abstraction result with respect to Levy-Longo Trees [24] relies on up-to techniques for bisimilarity. Precisely, given a relation $R$ on $\lambda$-terms that represents a tree bisimulation, one shows that the $\pi$-calculus encoding of $R$ is a $\pi$-calculus bisimulation up-to context and expansion. Expansion is a preorder that intuitively guarantees that a term is ‘more efficient’ than another one (Appendix D.3). 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 AL\(\pi\)

Full abstraction with respect to $\eta$-Eager-Tree equality also holds for Milner’s simplest encoding, namely $\forall$ (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 $sa(x)$.\(x(y)\) are abbreviations for $vx\left(\exists x\right)$.

**Theorem 5.1.** $M \equiv_{\eta} N$ iff $V\left[ M \right] \cong \text{AL}\pi\left[ V\left[ N \right] \right]$, 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 ($\cong_{\text{AL}\pi}$) as bisimilarity makes use of a different labelled transition system (Appendix C) 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 $\mathcal{I}$. Similar results however hold for the encoding $V$ into $\text{AL}\pi$.

### 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 $\mathcal{I}$ agents $A, B$ are in the contextual preorder, written $A \preceq_{\text{ctx}} \mathcal{I} B$, if $C[A] \cong \mathcal{I} C[B]$ for all contexts $C$. They are contextually equivalent, written $A \cong_{\text{ctx}} \mathcal{I} B$, if both $A \preceq_{\text{ctx}} \mathcal{I} B$ and $B \preceq_{\text{ctx}} \mathcal{I} A$ 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 $P \Rightarrow s$ if $P \Rightarrow \mu_1 P_1 \Rightarrow \mu_2 P_2 \cdots \Rightarrow \mu_n P_n$, for some processes $P_1, \ldots, P_n$.

**Definition 6.2.** Two $\mathcal{I}$ processes $P, Q$ are in the trace inclusion, written $P \lessdot \mathcal{I} Q$, if $P \Rightarrow s$ implies $Q \Rightarrow s$, for each trace $s$. They are trace equivalent, written $P \equiv_{\mathcal{I}} Q$, if both $P \lessdot \mathcal{I} Q$ and $Q \lessdot \mathcal{I} P$ hold.

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

**Theorem 6.3.** In $\mathcal{I}$, relation $\cong_{\text{ctx}} \mathcal{I}$ coincides with $\lessdot_{\mathcal{I}}$, and relation $\cong_{\text{ctx}} \mathcal{I}$ coincides with $\equiv_{\mathcal{I}}$.

### 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 $F$ is a pre-fixed point for $\leq_{\mathcal{I}P}$ of a system of equations $(\bar{X} = \bar{E})$ if $E[F] \leq_{\mathcal{I}P} F$; similarly, $F$ is a post-fixed point for $\leq_{\mathcal{I}P}$ if $F \leq_{\mathcal{I}P} E[F]$.

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

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

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 $\bar{F}$ is a pre-fixed point for $\leq_{\mathcal{I}P}$ of $E$, and $\bar{G}$ a post-fixed point, then $\bar{F} \leq_{\mathcal{I}P} \bar{G}$.

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 $E$ 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 $E'$ and $E$ where $E'$ extends $E$ with respect to $\leq_\tau$. Furthermore, suppose $E'$ is guarded and has a divergence-free syntactic solution. If $\bar{F}$ is a pre-fixed point for $\leq_\tau$ of $E'$, and $\bar{G}$ is a post-fixed point, then $\bar{F} \preceq_\tau \bar{G}$.

### 6.3 Full abstraction results

The preorder on $\lambda$-terms induced by the contextual preorder is $\eta$-eager normal-form bisimulation, $\leq_\eta$. It is obtained by imposing that $M \leq_\eta N$ 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 $\leq_\eta$ and its converse $\geq_\eta$.)

**Theorem 6.8** (Full abstraction on preorders). For any $M, N \in A$, we have $M \leq_\eta N$ iff $I(\langle M \rangle) \not\prec_{\leq_\eta} I(\langle N \rangle)$.

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 $M\eta N$ diverges and $N$ has an eager normal form. In this case, we use the following equation instead:

$$X_{M,N} \equiv (\bar{g}) \bar{I}(\langle \Omega \rangle).$$

(3)

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

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

Defining $I^\ast(\langle R_1 \rangle]$ and $I^\ast(\langle R_2 \rangle]$ as previously, we need to prove that $I^\ast(\langle R_1 \rangle] \leq_\tau E_R^\ast[I^\ast(\langle R_1 \rangle)]$ and $E_R^\ast[I^\ast(\langle R_2 \rangle)] \preceq_\tau I^\ast(\langle R_2 \rangle]$. The former result is established along the lines of the analogous result in Section 4.2: indeed, $I^\ast(\langle R_1 \rangle]$ is a solution of $E_R^\ast$ for $\approx$, and $\approx_\tau$ is coarser than $\approx$. For the latter, the only difference is due to equation (3), when $M\eta N$ and $N$ diverges but not $M$. In this case, we have to prove that $I(\langle \Omega \rangle] \preceq_\tau I(\langle N \rangle]$, which follows easily because the only trace of $I(\langle \Omega \rangle]$ is the empty one, hence $I(\langle \Omega \rangle)(p) \preceq_\tau P$ for any $P$.

**Corollary 6.9** (Full abstraction for $\preceq_{\leq_\tau}$). For any $M, N \in A$, $M \equiv_\eta N$ iff $I(\langle M \rangle) \not\prec_{\leq_\eta} I(\langle N \rangle]$.

### 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; as behavioural 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 $\lambda\eta$, so to control the capabilities of exported names. We have proved that, in both cases, the equivalence induced is Eager-Tree equality, modulo $\eta$ (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 Levy-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 Levy-Longo Trees and Böhm Trees [26].

For controlling capabilities, we have used $\lambda\eta\pi$. 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 essential in the completeness proofs with respect to Levy-Longo Trees and Böhm Trees for the call-by-name encodings. We have discussed the problems with call-by-value in Remark 4.9.)

For our encodings we have used the polyadic $\pi$-calculus; Milner’s original paper [16] used the monadic calculus (the polyadic $\pi$-calculus makes the encoding easier to read; it had not been introduced at the time of [16]). We believe that polydacity does not affect the results in the paper (the possibility of autoconcurrency breaks full abstraction of the encoding of the polyadic $\pi$-calculus into the monadic one, but autoconcurrency does not appear in the encoding of $\lambda$-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 [17]), appears more delicate, as we cannot rely on the usual notion of evaluation context.

Interpretations of $\lambda$-calculus into $\pi$-calculus appear related to game semantics [5, 8, 9]. In particular, for untyped call-by-name they both allow us to derive Böhm Trees and Levy-Longo Trees [10, 19]. To our knowledge, game semantics exist based on typed call-by-value [10, 19], and for (weak and strong) call-by-name, the equalities induced are those of Levy-Longo Trees and Böhm Trees [26]. To our knowledge, game semantics exist based on typed call-by-value, e.g., [2, 8], but not in the untyped case. In this respect, it would be interesting to see whether the relationship between $\pi$-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) of Université de Lyon) and Elica, ANR-14-CE25-0005, and the Université Franco-Italienne under the programme Vinci.

### References


A List of symbols for behavioural relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Description</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\equiv$</td>
<td>eager normal-form bisimilarity</td>
<td>(Definition 1.3)</td>
</tr>
<tr>
<td>$\equiv_\eta$</td>
<td>$\eta$-eager normal-form bisimilarity</td>
<td>(Definition 1.6)</td>
</tr>
<tr>
<td>$\leq_\sigma$</td>
<td>$\sigma$-barbed congruence in $L$</td>
<td>(Section 6.3)</td>
</tr>
<tr>
<td>$\leq$</td>
<td>(weak) bisimilarity</td>
<td>(Definition 1.10)</td>
</tr>
<tr>
<td>$\leq^\mu$</td>
<td>expansion</td>
<td>(Appendix B)</td>
</tr>
<tr>
<td>$\sim^\Lambda$</td>
<td>contextual equivalence in $\Lambda$</td>
<td>(Section 6.1)</td>
</tr>
<tr>
<td>$\sim^L$</td>
<td>contextual equivalence in $L$</td>
<td>(Section 6.1)</td>
</tr>
<tr>
<td>$\preceq^\mu$</td>
<td>contextual preorder in $\mu$</td>
<td>(Section 6.1)</td>
</tr>
<tr>
<td>$\preceq^\tau$</td>
<td>trace equivalence</td>
<td>(Section 6.1)</td>
</tr>
<tr>
<td>$\preceq^\tau$</td>
<td>trace inclusion</td>
<td>(Section 6.1)</td>
</tr>
</tbody>
</table>

Where $L$ is supposed to be a subcalculus of $\pi$, in the paper we have considered $\tau$ and $\Lambda\pi$.

B Operational semantics and the expansion preorder

B.1 Labelled Transition System

Transitions of $\pi$-calculus processes are of the form $P \xrightarrow{\mu} P'$, where

$$\mu := a(b) \mid \nu d \bar{a}(b) \mid \tau.$$ 

We abbreviate $\forall \nu \bar{a}(b)$ as $\bar{a}(b)$. The occurrences of $\bar{b}$ in $a(\bar{b})$ and those of $d$ in $\nu d \bar{a}(b)$ are bound; accordingly one defines the sets of bound names and free names of an action $\mu$, respectively written $bn(\mu)$ and $fn(\mu)$. The set of all the names appearing in $\mu$ (both free and bound) is written $n(\mu)$. Figure 2 presents the transition rules for the $\pi$-calculus.

Some standard notations for transitions: $\xrightarrow{\sigma}$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\xrightarrow{} = \xrightarrow{\sigma} = \xrightarrow{\tau}$ (the composition of the three relations). Moreover, $P \xrightarrow{\mu} P'$ holds if $P \xrightarrow{\mu_1} P_1$ or $P \xrightarrow{\mu_2} P_2$ or $P \xrightarrow{\mu_3} P_3$; similarly $P \xrightarrow{\mu} P'$ holds if $P \xrightarrow{\mu_1} P_1$ or $P \xrightarrow{\mu_2} P_2$ or $P \xrightarrow{\mu_3} P_3$. Finally, $P \xrightarrow{\tau} P'$ holds if there is $P'$ with $P \xrightarrow{\mu} P'$, and similarly for other forms of transitions.

B.2 Expansion

We define the expansion preorder, written $\preceq$, where $P \preceq Q$ intuitively means that $P$ and $Q$ have the same behaviour, and that $P$ may not be ‘slower’ (in the sense of doing more $\xrightarrow{\tau}$ transitions) than process $Q$.

**Definition B.1. Expansion, written $\preceq$, is defined as the largest relation $R$ such that $P R Q$ implies**

1. if $P \xrightarrow{\mu} P'$, then for some $Q', Q \xrightarrow{\mu} Q'$ and $P' R Q'$, and
2. if $Q \xrightarrow{\tau} Q'$, then for some $P', P \xrightarrow{\tau} P'$ and $P' R Q'$.

The converse of $\preceq$ is written $\succeq$.

As usual, expansion is extended to abstractions by requiring ground instantiation of the parameters: $F \preceq F'$ if $F(\bar{a}) \approx F'(\bar{a})$, where $\bar{a}$ are fresh names of the appropriate sort. Expansion is finer than bisimilarity, i.e., $\preceq \subset \equiv$. 

---

Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi
Figure 2. Labelled Transition Semantics for the π-calculus

Figure 3. The modified labelled transition system for ALπ

C ALπ: Operational Semantics

We present here the results which make it possible for us to apply the unique-solution technique to ALπ. The main idea is to exploit a characterisation of barbed congruence as ground bisimilarity.

However, to obtain this [15], ground bisimilarity has to be set on top of a non-standard transition system, specialised to ALπ.

The Labelled Transition System (LTS) is produced by the rules in Figure 3; these modify ordinary transitions (the ⤠파인 relation) by adding static links a ⤦ b, which are abbreviations defined thus:

\[ a \dashv b \overset{def}{=} !(a(x),b(x)) \]

(We call them static links, following the terminology in [15], so to distinguish them from the links a ⤦ b used in 1π, whose definition makes use of recursive process definitions -- static links only need replication.)

Notations for the ordinary LTS (\(\rightarrow\)) are transported onto the new LTS (\(\overset{\mu}{\rightarrow}\)), yielding, e.g., transitions \(\mu\) and \(\overset{\mu}{\rightarrow}\).

We write \(\approx\) for (ground) bisimilarity on the new LTS (defined as \(\approx\) in Definition 1.10, but using the new LTS in place of the ordinary one). Barbed congruence in ALπ, \(\approx_{AL\pi}\), is defined as by Definition 1.8 (on \(\tau\)-transitions, which are the only transitions needed to define \(\approx_{AL\pi}\), the new LTS and the original one coincide).

We present the definition of asynchronous (ground) bisimilarity, which is used in [15] to derive a characterisation of barbed congruence (asynchrony is needed because the calculus is asynchronous, and barbed congruence observes only output actions).

Definition C.1 (Asynchronous bisimilarity). Asynchronous bisimilarity, written \(\approx_{as}\), is the largest symmetric relation \(R\) such that \(PRQ\) implies

- if \(P \overset{\mu}{\rightarrow} P'\) and \(\mu\) is not an input, then there is \(Q'\) s.t. \(Q \overset{\mu}{\rightarrow} Q'\) and \(P'\overset{\mu}{\rightarrow}Q'\), and
- if \(P \overset{a\pi}{\rightarrow} P'\), then either \(Q \overset{a\pi}{\overrightarrow{\rightarrow}} Q'\) and \(P' \overset{a}{\overrightarrow{\rightarrow}Q'}\) for some \(Q'\), or \(Q \overset{\pi}{\overrightarrow{\rightarrow}Q'}\) and \(P' \overset{\pi}{\overrightarrow{\rightarrow}Q'}\) for some \(Q'\).

Theorem C.2 ([15]). On image-finite up to \(\approx_{AL\pi}\) processes, relations \(\approx_{as}\) and \(\approx_{AL\pi}\) coincide.

To apply our technique of unique solutions of equations it is however convenient to use (synchronous) bisimilarity. The following result allows us to do so:

Theorem C.3. On image-finite up to \(\approx_{AL\pi}\) processes that have no free inputs, relations \(\approx_{as}\) and \(\approx_{as}\) coincide.

For any \(M \in \Lambda\) and \(p\), process \(\llbracket M \rrbracket(\pi)p\) is indeed image-finite up to \(\approx\) and has no free input. We also exploit the fact that \(\approx_{as}\) is a congruence relation in ALπ. The property in Theorem C.3 is new -- we are not aware of papers in the literature presenting it. It is a consequence of the fact that, under the hypothesis of the theorem, and with a ground transition system, the only input actions in processes that can ever be produced are those emanating from the links, and two tested processes, if bisimilar, must have the same sets of (visible) links.

Part (2) of Theorem 1.11 (Section 1.3) then follows from Theorems C.2 and C.3.

D Full Abstraction of the encoding in 1π

We present here some additional material regarding the structure of the proofs of the results in Section 4 (though we do not intend to provide full proofs below).

D.1 Validity of βς-reduction

The goal of this section is to prove Lemma 4.1, that is, validity of \(\beta_\varsigma\)-reduction. First, we recall some useful properties of links [23].

The first one is a form of composition, expressed in terms of the expansion preorder (Definition B.1).

Lemma D.1. \(\forall b (a \circ b | b \circ c) \geq a \circ c\).

Definition D.2. Given a value \(V\), we define \(\overline{I}_\pi[\overline{V}](z)\) as follows:

1. If \(V = x\), then \(\overline{I}_\pi[\overline{V}](z) = z \circ x\).
2. If \(V = \lambda x. M\), then \(\overline{I}_\pi[\overline{V}](z) = !z(x,q).\overline{I}[\overline{M}](q)\).

We observe that for any value \(V\), we have:

\(\overline{I}[\overline{V}](p) = \overline{P}(z).\overline{I}[\overline{V}](z)\).

The next lemma shows that, on the processes obtained by the encoding into 1π, links behave as substitutions. We recall that \(p,q\) are continuation names, whereas \(x,y\) are variable names.

Lemma D.3. We have:

1. \(\forall x \(\overline{I}[\overline{M}(p) \mid x \circ y] \geq \overline{I}[\overline{M}(y/x)](p)\)
2. \(\forall p \(\overline{I}[\overline{M}(p)](p) \circ q \geq \overline{I}[\overline{M}](q)\)
3. \(\forall y \(\overline{I}[\overline{V}](y) \mid x \circ y] \geq \overline{I}[\overline{V}](x)\)
Proof. Laws 1 and 2 are proved by induction on \( M \), using algebraic reasoning and Lemma D.1. Law 1 is needed to show law 2. Law 3 can be derived from laws 1 and 2, by case analysis on \( V \).

We sometimes use Lemma D.3 with \( \equiv \) in place of \( \geq \).

**Lemma D.4.** \( I(M[V/x]) \geq I(M[V/x]) \), for all \( M \) and value \( V \).

**Proof.** We first show, using algebraic reasoning, that:

\[
I(M[V/x]) \geq I(M[V/x])
\]

Then, with a straightforward induction on \( M \), we derive

\[
\forall x (I(M)[p] \models I(M[V/x])[p]).
\]

Now Lemma 4.1, stating that \( M \rightarrow N \) implies \( I(M) \geq I(N) \) (the validity of \( \beta \),-reduction), follows from Lemma D.4 and the congruence properties of \( \equiv \). A stronger property, stated using expansion, actually holds:

\[
M \rightarrow N \text{ implies } I(M) \geq I(N).
\]

**D.2 Soundness proof**

To prove soundness we need to establish an operational correspondence for the encoding. For this it is easier to relate \( \lambda \)-terms and \( \Pi \)-terms via an optimised encoding \( O \), presented in Figure 4. In the figure we assume that rules \( \text{var-val} \) and \( \text{abs-val} \) have priority over the others; in other words, in rules \( \text{var-app} \), \( \text{app-val} \), and \( \text{app} \), terms \( M \) and \( N \) should not be values. In the optimised encoding, we remove some initial communications in the encoding of applications. To achieve this, the encoding of an application goes by a case analysis (4 cases) on the occurrences of values in the subterms.

**Lemma D.5.** We have:

\[
O(C_x[xv]) \geq \tau(x,q). (O_C[v](z) \mid q(y). O[C_y(y)](p)).
\]

**Proof.** By induction on the evaluation context \( C_x \).

The following lemma, relating the original and the optimised encoding, allows us to use the latter to establish the soundness of the former. The proof uses a few simple algebraic laws.

**Lemma D.6.** \( I(M) \geq O(M) \), for all \( M \in \Lambda \).

In the lemma below, recall that we identify processes or transitions that only differ in the choice of the bound names.

**Lemma D.7 (Operational correspondence).** For any \( M \in \Lambda \) and fresh \( p \), process \( O(M)(p) \) has exactly one immediate transition, i.e.,

\[
O(M)(p) \rightarrow_p P \text{ and } O(M)(p) \overset{\mu}{\rightarrow} P' \text{ imply } \mu = \mu' \text{ and } P = P' \text{ (as usual, up to } a\)-conversion), and exactly one of the following clauses holds:

1. \( O(M)(p) \rightarrow P \text{ and } P = \text{a value, with } P = O_C[M](y); \)
2. \( O(M)(p) \overset{\tau(x,q)}{\rightarrow} P \text{ and } M = C_x[xv] \) and

\[
P \geq O_C[V](z) \mid q(y). O[C_y(y)](p);\]
3. \( O(M)(p) \overset{\tau}{\rightarrow} P \text{ and there is } N \text{ with } M \rightarrow N \text{ and } P \geq O(N)(p).\)

**Proof.** By induction on \( M \). We use Lemmas 4.1, D.1, and D.3.

**Lemma D.8.** If \( O(M)(p) \rightarrow P \text{ and } \mu \neq \tau \), then \( M \) admits an eager normal form \( M' \) such that \( O(M')(p) \overset{\mu}{\rightarrow} P \).

**Proof.** By induction on the length of the reduction \( O[M](p) \overset{\mu}{\rightarrow} P \). If \( O[M](p) \overset{\mu}{\rightarrow} P \text{ and } \mu \neq \tau \), by Lemma D.7, \( M \) is an eager normal form; otherwise, there is \( P' \) such that \( O[M](p) \overset{\tau}{\rightarrow} P' \overset{\mu}{\rightarrow} P \); by Lemma D.7, there is \( N \) such that \( M \rightarrow N \), and \( P' \geq O[N](p) \). Therefore \( O[N](p) \rightarrow Q \), where \( Q \geq P \) and the reduction is shorter. By the induction hypothesis, \( N \) admits an eager normal form \( M' \), which is also an eager normal form of \( M \), with \( O(M')(p) \overset{\mu}{\rightarrow} Q \geq Q \).

The following lemma allows us to decompose an equivalence between two parallel processes. This result is used to handle equalities of the form \( I(C_x[xv]) \equiv I(C_x[xv']) \), in order to deduce equivalence between \( V \) and \( V' \) on the one hand, and between \( C_x[y] \) and \( C_x[y] \) on the other.

**Lemma D.9.** Suppose that \( a \) does not occur free in \( Q \) or \( Q' \), and one of the following holds:

1. \( a(x). P \mid Q \approx a(x). P' \mid Q' \).
2. \( \lambda a(x). P \mid Q \approx \lambda a(x). P' \mid Q' \).

Then we also have \( Q \approx Q' \).

**Proof.** Since \( a \) does not appear free in \( Q \) or \( Q' \), neither of those processes can perform an action in which \( a \) occurs free (here we rely on the labelled translation system being ground). Thus the bisimulation game for \( Q \) and \( Q' \) can be derived from that for \( a(x). P \mid Q \) and \( a(x). P' \mid Q' \) or that for \( \lambda a(x). P \mid Q \) and \( \lambda a(x). P' \mid Q' \).

We now show that the only \( \lambda \)-terms whose encoding is bisimilar to \( x[V] \) reduce either to \( x \), or to a (possibly infinite) \( \eta \)-expansion of \( x \).

**Lemma D.10.** If \( V \) is a value and \( x \) a variable, \( O_C[V] \approx O_C[x] \) implies that either \( V = x \) or \( V = \lambda x. M \), where the eager normal form of \( M \) is of the form \( C_x[xv'] \), with \( O_C[V'] \approx O_C[z] \) and \( O_C[C_y(y)] \approx O_C[y] \) for any fresh \( y \).

**Proof.** By definition, \( O_C[v](p) = \tau(y)(z,q). O[C_y(y)](q) \mid q(y). O[C_y(y)](p) \).

Therefore \( O_C[M](q) \geq \tau(z,q'). (z' \rightarrow z \mid q' \rightarrow q' \rightarrow q'. \)

by Lemmas D.8 and D.7. \( M \) has an eager normal form \( C_x[xv'] \). We have, using Lemma D.5:

\[
O[C_x[xv']](q) \approx \tau(z,q'). (O[C_x[xv']](z') \mid q'(y). O[C_y(y)](q)).
\]

since

\[
O[C_x[xv']](z') \mid q'(y). O[C_y(y)](q) \approx z' \rightarrow z \mid q' \rightarrow q'.
\]

Then, \( z' \) is not free in \( q'(y). O[C_y(y)](q) \) and \( q' \) is not free in \( O[C_x[xv']](z') \). Furthermore, \( O[C_x[xv']](z') \) is prefixed by an input on \( z' \). By applying Lemma D.9 twice, we derive

\[
O[C_x[xv']](z') \approx z' \rightarrow z \mid q'(y). O[I[y]](q) \approx q' \rightarrow q' \rightarrow q.
\]

By definition, \( z' \rightarrow z = \tau(z,q). (x \rightarrow y) \), so

\[
O[C_x[xv']](z) = O[C_x[xv']](z')
\]
Eager Functions as Processes

LICS’18, July 9–12, 2018, Oxford

where $O_Y$ is thus defined:

\[ O_Y[M] \equiv (y) ! (y(x), O_Y[M](q)) \]

Moreover, in rules VAL-APP and APP-VAL, $M$ is not a value; in rule APP $M$ and $N$ are not values.

**Figure 4.** Optimized encoding into $\lambda r$

Also

\[ q'(y), O_Y[C_e[y]](q) \approx q' \gg q \]

\[ = q'(y), \overline{q}(y'). (y' \gg y) \]

By playing the bisimilarity game over the previous processes, we derive

\[ O_Y[C_e[y]](q) \approx \overline{q}(y'). (y' \gg y) \]

\[ = O_Y[y](q) \]

\[ \square \]

**Lemma D.11.** $O[[M]](p) \approx O[[N]](p)$ iff $M \triangleleft p$.

**Proof.** 1. Suppose $O[[M]](p) \neq (p) 0$. Then $M \xrightarrow{\mu} P$ for some $\mu \neq r$, and by Lemma D.8, $M$ has an eager normal form (hence $M$ does not diverge).

2. Assume now $O[[M]](p) \approx (p) 0$. By Lemma D.7, $O[[M]](p) \xrightarrow{\mu} P$ for some $P$, and there is $N$ such that $M \xrightarrow{\mu} N$, and $N \approx p \approx (p) 0$. With this property, using coinduction we derive $M \triangleleft p$.

\[ \square \]

We can now present the proof of soundness.

**Proof of Theorem 4.2.**

Let $R \equiv \{ (M, N) \mid O[[M]] = O[[N]] \}$; we show that $R$ is an $\eta$-eager normal form bisimulation, and conclude by Lemma D.6.

Assume $O[[M]] = O[[N]]$.

1. If $M \uparrow$ or $N \uparrow$, by Lemma D.11, $I[[M]](p) \approx I[[N]](p) \approx 0$, hence both diverge.

2. Otherwise, $M$ and $N$ have eager normal forms $M'$ and $N'$; i.e., $M \triangleleft M'$ and $N \triangleleft N'$. Therefore by Lemma D.6 and validity of $\beta_\nu$-reduction, $O[[M']] \approx O[[N']]$. Since $M'$ is in eager normal form, by Lemma D.7, either $O[[M']] = O[[N']]$ or $\overline{P}(y)$, $O[[M']] \xrightarrow{\mu \rightarrow P}$, and likewise for $N'$. This gives rise to two cases:

a. $M' = C_e[xV]$ and $N' = C_e[xV']$, and

\[ O_Y[[V][\nu]](q), O_Y[[C_e[y]]](p) \approx O_Y[[V'][\nu]](q), O_Y[[C_e[y]]](p) . \]

b. Also, $z'$ does not appear free in $O_Y[[V][\nu]](z)$ or $O_Y[[V'][\nu]](z)$, hence by Lemma D.9

\[ O_Y[[V]] \approx O_Y[[V']] . \]

D.3 Completeness proof

**Systems $\mathcal{E}_R$ and $\mathcal{E}'_R$.** We provide the full description of the systems of equations $\mathcal{E}_R$ (Figure 5) and $\mathcal{E}'_R$ (Figure 6). There, $\overline{y}$ is assumed to be the ordering of $v_N, M$. In $\mathcal{E}'_R$, we write $\overline{y}'$ or $\overline{y}''$ for the free variables of the terms indexing the corresponding equation variable.

For both systems, we give all equations needed to handle $\equiv_R$. As explained in Section 4.2, the systems to handle $\equiv_R$ are obtained by omitting some equations (precisely, the last two equations in Figures 5 and 6).

**Proof of Lemma 4.3.** The remainder of this section is devoted to the proof of Lemma 4.3; we recall its statement:

If $R$ is an eager normal form bisimulation, then $I[[R]]$ and $I[[\overline{R}]]$ are solutions of $\mathcal{E}_R$.

Before explaining how $R$ yields solutions of the system, we prove law (2) from Section 4.2.

**Lemma D.12.** If $C_e$ is an evaluation context, $V$ is a value, $x$ is a name and $z$ is fresh in $C_e$, then

\[ I[[C_e[xV]]] \approx I[[\nu(z, C_e[z])(xV)]] . \]
\[ M \uparrow \text{ and } N \uparrow \quad X_{MN} = (\widetilde{y}, p) I I [\Omega] \]

\[ M \uparrow x \text{ and } N \uparrow x \quad X_{MN} = (\widetilde{y}, p) I I [x] \]

\[ M \uparrow \lambda x. M' \text{ and } N \uparrow \lambda x. N' \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. X_{M'N'}] \]

\[ M \uparrow \text{ and } N \uparrow \lambda x. N' \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. X_{M'N'}] \]

\[ M \uparrow \lambda x. C_e[xV] \text{ and } N \uparrow \lambda x. C'_e[xV'] \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. (\lambda w. X_{C_e[w]}(xV, z)) H \langle \lambda x. \lambda w. C'_e[w] \rangle (xV, z)] \]

\[ M \uparrow x, N \uparrow \lambda z. N', N' \uparrow \lambda z. C_e[xV'] \quad X_{MN} = (\widetilde{y}, p) I I [\lambda z. (\lambda w. X_{C_e[w]}(xV, z)) H \langle \lambda x. \lambda w. C'_e[w] \rangle (xV, z)] \]

\[ M \uparrow \lambda z. M', M' \uparrow \lambda z. C_e[xV'], N \uparrow x \quad X_{MN} = (\widetilde{y}, p) I I [\lambda z. (\lambda w. X_{C_e[w]}(xV, z)) H \langle \lambda x. \lambda w. C'_e[w] \rangle (xV, z)] \]

**Figure 5.** System \( E_R \) of equations (the last two equations are included only when considering \( \varepsilon = _\eta \))

\[ \quad M \uparrow \text{ and } N \uparrow \quad X_{MN} = (\widetilde{y}, p) I I [\Omega] \]

\[ \quad M \uparrow \text{ and } N \uparrow \lambda x. N' \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. M''] \]

\[ \quad M \uparrow \lambda x. M' \text{ and } N \uparrow \lambda x. N' \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. M''] \]

\[ \quad M \uparrow \lambda x. C_e[xV] \text{ and } N \uparrow \lambda x. C'_e[xV'] \quad X_{MN} = (\widetilde{y}, p) I I [\lambda x. (\lambda w. X_{C_e[w]}(xV, z)) H \langle \lambda x. \lambda w. C'_e[w] \rangle (xV, z)] \]

\[ \quad M \uparrow x, N \uparrow \lambda z. N', N' \uparrow \lambda z. C_e[xV'] \quad X_{MN} = (\widetilde{y}, p) I I [\lambda z. (\lambda w. X_{C_e[w]}(xV, z)) H \langle \lambda x. \lambda w. C'_e[w] \rangle (xV, z)] \]

**Figure 6.** System \( E_R' \) of equations (the last two equations are included only when considering \( \varepsilon = _\eta \))

\[ \quad (\widetilde{y}, p) I I [M](p) \quad \text{and} \quad (\widetilde{y}, p) I I [N](p) \]

Proof: By induction on the evaluation context \( C_e \). When \( C_e = [\cdot] \), we show that \( I I [xV] = I I [I I (xV)] \) by algebraic reasoning. The other cases can be handled similarly. \( \square \)

We recall that, if \( y = fv(M, N) \), then \( (\widetilde{y}, p) I I [M](p) \) are closed abstractions.

Proof of Lemma 4.3. We only show the property for \( R_1 \), the case for \( R_2 \) is handled similarly.

We have to show that, given \((M, N) \in R, \) we have \( (\widetilde{x}, q) I I [M] \approx E_{MN} [I I [R]] \).

- If \( M \downarrow \), we use Lemma D.11, which gives us \( (\widetilde{x}, q) I I [M] \approx (\widetilde{x}, q) I I [\Omega] \).

- If \( M \uparrow \lambda x. C_e[xV] \), we have to show that:

\[ I I [M] \approx I I [(\lambda x. C_e[z])(xV)] \]

By Lemma 4.1, \( I I [M] \approx I I [C_e[xV]] \); we then conclude by Lemma D.12.

- If \( M \uparrow \lambda x. M' \) (and \( N \) also reduces to an abstraction), then:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(x, q) \mid I I [M')(q) \approx I I [M](\widetilde{y}, p) \} \]

(by Lemma 4.1).

- If \( M \downarrow x \) (and \( N \) \( \downarrow x \)), again:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(x)(p) \approx I I [M](\widetilde{y}, p) \} \]

(by Lemma 4.1).

Completion for \( \varepsilon = _\eta \). To move from \( \varepsilon \) to \( \varepsilon = _\eta \), we consider an \( \eta \)-eager normal-form bisimulation \( \mathcal{R} \), and we use the full systems of equations \( E_R \) and \( E_R' \).

The proof follows the same lines as for \( \varepsilon \), the only notable difference being that more cases are to be considered to show that \( I I [\mathcal{R}_1] \) and \( I I [\mathcal{R}_2] \) are solutions of \( E_R \).

We first prove the following lemma, which validates \( \eta \)-expansions for the encoding.

**Lemma D.13.** \( I I [\lambda y. xy] \approx I I [x] \).

Proof. Using usual algebraic laws and the definition of links. \( \square \)

The proof that \( I I [\mathcal{R}_1] \) is solution follows closely the proof of Lemma 4.3, with the two additional cases to be considered:

- If \( M \downarrow x \) and \( N \downarrow _\lambda y. N' \), then:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(x, q) \} \approx \{z(x, q) \} \]

(by Lemma D.12).

- If \( M \uparrow \lambda y. M' \) (and \( N \) becomes a fresh variable), then:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(y, q) \} \approx \{z(y, q) \} \]

(by Lemma D.12).

- If \( M \downarrow xy \) (and \( N \) becomes a fresh variable), then:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(y, q) \} \approx \{z(y, q) \} \]

(by Lemma D.12).

- If \( M \uparrow \lambda y. M' \) \( \uparrow C_e[xV] \) and \( N \downarrow x \), then:

\[ E_{MN} [I I [R]](\widetilde{y}, p) = \{z(y, q) \} \approx \{z(y, q) \} \]

(by Lemma D.12).