Final semantics for the π-calculus*

F. Honsell, M. Lenisa
Università di Udine
Via delle Scienze 208, Udine, Italy. {honsell,lenisa}@dimi.uniud.it

U. Montanari, M. Pistore
Università di Pisa
Corso Italia 40, Pisa, Italy. {ugo,pistore}@di.unipi.it

Abstract

In this paper we discuss *final semantics* for the π-calculus, a process algebra which models systems that can dynamically change the topology of the channels. We show that the final semantics paradigm, originated by Aczel and Rutten for CCS-like languages, can be successfully applied also here. This is achieved by suitably generalizing the standard techniques so as to accommodate the mechanism of *name creation* and the behaviour of the binding operators peculiar to the π-calculus. As a preliminary step, we give a *higher order* presentation of the π-calculus using as metalanguage LF, a *logical framework* based on typed λ-calculus. Such a presentation highlights the nature of the binding operators and elucidates the rôle of free and bound channels. The final semantics is defined making use of this higher order presentation, within a category of *hypersets*.

Keywords

Semantics, process calculi, coalgebras, logical frameworks, hypersets

1 INTRODUCTION

The π-calculus [12, 11] is a process algebra which models communicating systems that can dynamically change the topology of the channels. This is obtained by allowing (channel) names to appear as values in communications. The possibility of reconfiguring process adjacency gives to the π-calculus a richer expressive power than CCS: for instance, π-calculus can be used to model object oriented languages, and higher-order communications can be encoded in it.

*Research supported in part by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione* and Esprit Working Groups *CONFER2* and *TYPES*.

However, this possibility of communicating names, and hence of dynamically modifying the scope of local names, leads to new problems when one tries to extend to the π-calculus the techniques which have been developed for CCS-like languages. In this paper we address the problem of extending to the π-calculus the *final-semantics* approach originally introduced by Aczel, Rutten and Turi [2, 15, 14] for CCS-like languages and further generalized to λ-calculus [8] and higher order concurrent imperative languages [9].

The gist of the *final semantics paradigm* [2] is to view the *interpretation* function from *syntax* to *semantics* as a *final* mapping in a suitable category. To this end the semantics has to be construed as a *terminal coalgebra* for a suitable functor F and the syntax has to be cast in the form of an F-coalgebra. In a sense, this approach is driven by the operational semantics of the language, because it is the semantics which determines the structure of the functor F. This is dual to the syntax-driven approach of *initial semantics*, where syntax is construed as an initial F-algebra and the semantics is defined as an F-algebra. As Rutten [14] has clearly explained, the duality between final semantics and initial semantics can be pushed very far and it is rather fruitful. By way of example we just mention a few intriguing correspondences: *congruences* are dual to *bisimulations*, so that *induction* (i.e. identity is the least congruence) is dual to *coinduction* (i.e. identity is the largest bisimulation); *inductively defined* functions (i.e. unique maps out of the initial algebra) are dual to *coiterative* functions (i.e. unique maps into the final algebra).

More specifically, the final semantics paradigm consists in representing the operational semantics of a programming language as a *syntactical F-coalgebra*, for a suitable functor F, in such a way that the appropriate behavioral equivalence (e.g. bisimilarity) can be characterized as the equivalence induced on terms (processes) by the unique morphism from the syntactical coalgebra to the *final F-coalgebra*. Final semantics is a fruitful way of understanding the correspondence between syntax and operational semantics. First of all, it naturally validates *coinductive* proof techniques for establishing behavioral equivalences, as well as other properties, and it immediately allows to define semantical operators by *corecursion*. But more than this, it allows to phrase such proofs by coinduction in a more principled and uniform way using the notion of F-coalgebra (see Section 6). And furthermore, it allows to view corecursion and coinduction as two aspects of the same phenomenon.

In order to extend the final semantics approach to the π-calculus, we have found it convenient to give first a *higher order* presentation of it, using as metalanguage a logical framework based on typed λ-calculus, such as the Edinburgh Logical Framework *LF* [7]. This *LF* presentation provides useful insights into the nature of the various binding operators of the π-calculus, thus allowing to focus on the uses of free and bound names. Capitalizing on this, one can overcome many of those problematic aspects of the π-calculus, which do not allow for a direct reuse of the techniques in [2, 15]. The proposed presentation-encoding of the π-calculus can be proved to be *faithful* to the original system using standard techniques. It can be viewed also as the specification of a *derivation editor* for the π-calculus, in any of the various computer assisted proof development environments based on constructive type theory such as *Coq* [3].

Utilizing the *LF* presentation, coalgebras are defined for the *strong* and for the *weak* semantics of the π-calculus. Strong and weak *late* bisimilarities are the equivalences induced by finality on the two coalgebras. In this paper we consider only the late semantics, but our techniques can be easily extended to the *early* [13] and *open* [16] semantics, see [10] for details. At the price of a slight complication, one can deal also with strong and weak *congruences*; we illustrate only the first case.

We give the final semantics within a category $Class^*$ [2, 15] of classes of hypersets, i.e. non-wellfounded sets satisfying the antifoundation axiom X_1 of [6] (aka AFA [1]), which asserts that the Universe of sets is a final-coalgebra for the functor $\mathcal{P}(\ )$. We use this setting because it is rather natural and moreover, as is the case for CCS [2], it seems that a set-theoretic setting is essential in dealing with the non-continuous character of weak semantics. Of course one could also work in the category of wellfounded sets Set, explaining away non-wellfounded sets at the price of some extra-coding and replacing identity with isomorphism.

Two versions of denotational semantics have been recently defined for the π-calculus in [17] and [5]. However, these approaches just work for *strong* semantics, and, as far as we know, no denotational semantics have been defined for the *weak* case. From this perspective, the final characterization of the weak semantics is particularly interesting.

The paper is organized as follows. In Section 2 we review the standard syntax and operational semantics of the π-calculus; in Section 3 we show how a convenient presentation of the operational semantics of the π-calculus can be given using *LF* as metalanguage. In Section 4 we present the definitions of strong and weak late bisimulations, and of late strong congruence. In Section 5 final characterizations of these semantics are obtained. Applications are discussed in Section 6. Concluding remarks appear in Section 7.

2 THE π-CALCULUS

In this section we review the ordinary syntax and operational semantics of (late, monadic) π-calculus. More details can be found in [12, 11].

Given an infinite set $\mathcal{N}$ of *names*, ranged over by $x, y, z \ldots$, the π-calculus *processes*, denoted by $p, q, r \ldots$, are defined by the syntax:

$$p ::= 0 \mid \tau.p \mid x(y).p \mid \bar{x}y.p \mid p|q \mid p{+}q \mid !p \mid (\nu x)p \mid [x{=}y]p.$$

The occurrences of y in $x(y).p$ and $(\nu y)p$ are bound; *free names*, $fn(p)$, are defined as usual. If $\sigma : \mathcal{N} \to \mathcal{N}$, we denote with $p\sigma$ the process p whose free names have been replaced according to substitution σ (possibly with α-

Table 1 Late operational semantics

$$TAU: \tau.p \xrightarrow{\tau} p \qquad OUT: \bar{x}y.p \xrightarrow{\bar{x}y} p \qquad IN: x(y).p \xrightarrow{x(y)} p$$

$$SUM^l: \frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad PAR^l: \frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q} \text{if } bn(\alpha) \cap fn(q) = \emptyset$$

$$COM^l: \frac{p \xrightarrow{\bar{x}y} p' \quad q \xrightarrow{x(z)} q'}{p|q \xrightarrow{\tau} p'|q'\{y/z\}} \qquad CLOSE^l: \frac{p \xrightarrow{\bar{x}(y)} p' \quad q \xrightarrow{x(y)} q'}{p|q \xrightarrow{\tau} (\nu y)(p'|q')}$$

$$RES: \frac{p \xrightarrow{\alpha} p'}{(\nu x)p \xrightarrow{\alpha} (\nu x)p'} \text{if } x \notin n(\alpha) \qquad OPEN: \frac{p \xrightarrow{\bar{x}y} p'}{(\nu y)p \xrightarrow{\bar{x}(y)} p'} \text{if } x \neq y$$

$$BANG: \frac{p|!p \xrightarrow{\alpha} p'}{!p \xrightarrow{\alpha} p'} \qquad COND: \frac{p \xrightarrow{\alpha} p'}{[x{=}x]p \xrightarrow{\alpha} p'}$$

$$EQ: \frac{p \equiv p' \quad p' \xrightarrow{\alpha} q' \quad q' \equiv q}{p \xrightarrow{\alpha} q}$$

conversions to avoid name clashing); we denote with $\{y_1/x_1 \cdots y_n/x_n\}$ the substitution that maps x_i into y_i for $i = 1, \ldots, n$, and which is the identity on the other names. Processes which differ by α-conversion, $\equiv$, are identified.

The *actions* that a process can perform, ranged over by α, are defined by the following syntax:

$$\alpha ::= \tau \mid x(z) \mid \bar{x}y \mid \bar{x}(z).$$

Names x and y are *free names* of α, $(fn(\alpha))$, whereas z is a *bound name* $(bn(\alpha))$; moreover $n(\alpha) = fn(\alpha) \cup bn(\alpha)$. The transitions for the *late operational semantics* are defined by the rules of Table 1 (rules SUM^r, PAR^r, COM^r and $CLOSE^r$ have been omitted).

In transition $p \xrightarrow{x(y)} p'$ (resp. $p \xrightarrow{\bar{x}(y)} p'$), name y represents a reference in p' for the place where the received name of the input will go (resp. where the private name which is emitted occurs). Since y is a reference, it has to be a *fresh* name, i.e., a name which is different from all the other names of p'. The side-conditions which appear in the rules PAR and $OPEN$ are required to enforce this. These transitions are called *bound transitions* (in contrast, τ- and $\bar{x}y$-transitions are called *free transitions*).

3 AN *LF* ENCODING OF THE π-CALCULUS

In this section we reformulate the syntax and the operational semantics of the π-calculus using as metalanguage the Edinburgh Logical Framework *LF* [7]. Logical Frameworks are general *specification languages* for formal systems where one can express uniformly all the features and aspects of an arbitrary system, e.g.: syntactic categories, variables, syntactic constructors, binding operators, substitution mechanisms, judgements, rules, derivations, etc. Encodings in Logical Frameworks readily provide "derivation editors" for the "object logic", given an editor for the metalanguage.

LF is based on a dependent typed λ-calculus and it exploits the well known "judgements (formulæ) as types", "proofs (derivations) as λ-terms" paradigm. *LF* allows also a smooth treatment of binding operators in terms of *higher order syntax* à la Church. More specifically, *LF* is a system for deriving typing assertions of the shape $\Gamma \vdash P : Q$, whose intended meaning is "in the environment Γ, P is classified by Q". Three kinds of entities are involved, i.e. *terms* (ranged over by M, N), *types* and *typed valued functions* (ranged over by A, B), and *kinds* (ranged over by K). Types are used to classify terms, and kinds are used to classify types and typed valued functions. These entities are defined by the following abstract syntax:

$$
\begin{array}{rcl}
M & ::= & x \mid MN \mid \lambda x : A.M \\
A & ::= & X \mid AM \mid \Pi_{x:A}.B \mid \lambda x : A.B \\
K & ::= & Type \mid \Pi_{x:A}.K.
\end{array}
$$

Π is the *dependent type* constructor. Intuitively $\Pi_{x:A}.B(x)$ denotes the type of those functions, f, whose domain is A and whose values belong to a codomain *depending* on the input, i.e. $f(a) \in B(a)$, for all $a \in A$. Hence, in *LF*, $A \to B$ is just notation abbreviating $\Pi_{x:A}.B$, when x does not occur free in B ($x \notin FV(B)$). In the "propositions-as-types" analogy the dependent product type $\Pi_{x:A}.B(x)$ represents the proposition $\forall x : A.\ B(x)$.

Environments are lists of pairs consisting of a variable and its type or kind.

The *LF* specification of a formal system is given by a particular environment, called *signature*, determined according to the following methodology. The *syntactic categories* of the abstract syntax language are encoded as type-variables, *syntactic constructors* are encoded as term-variables of the appropriate type, *judgements* are encoded as variables of "typed-valued function" kind, and *rules* are encoded as term-variables of the appropriate types. Hence, derivations of a given assertion are represented as terms of the type corresponding to the assertion in question, so that *checking* the correctness of a derivation amounts just to *type-checking*. Thus, *derivability* of an assertion corresponds to the *inhabitation* of the type corresponding to that assertion, i.e. the derivability of a term of that type.

Metalanguage variables play also the rôle of "object logic" variables so that *schemata* are represented as λ-abstractions and *instantiation* amounts to β-reduction. Hence *binding operators* are encoded as *higher order* variables, being viewed as ranging over schemata.

3.1 The LF-signature Σ_π of the π-calculus

The **syntactic categories** for names, labels and processes are encoded by the following types:

$$\begin{array}{lcl} name & : & Type \\ label & : & Type \\ proc & : & Type \end{array}$$

Syntactic constructors are defined for labels and processes. Functional types are used for constructors, e.g: the constructor *out* for output requires two names to generate a label and so it is represented by assigning to *out* type $name \to name \to label$.

- Constructors for *label*:

$$\begin{array}{lcl} \tau & : & label \\ in & : & name \to name \to label \\ out & : & name \to name \to label \end{array}$$

- Constructors for *proc*:

$$\begin{array}{lcl} 0 & : & proc \\ \tau_{pref} & : & proc \to proc \\ in_{pref} & : & name \to (name \to proc) \to proc \\ out_{pref} & : & name \to name \to proc \to proc \\ \mid & : & proc \to proc \to proc \\ + & : & proc \to proc \to proc \\ ! & : & proc \to proc \\ \nu & : & (name \to proc) \to proc \\ [\,=\,] & : & name \to name \to proc \to proc \end{array}$$

Notice that labels are defined only for free transitions: we will see below how bound transitions, such as bound output, are represented by giving a higher order type to the corresponding transition judgment. We shall freely pretty-print $in(x, y)$ as xy and $out(x, y)$ as $\bar{x}y$.

Notice also how π-calculus binders are expressed in LF making use of suitable abstractions. E.g.: in the case of the input prefix, in order to obtain a process, a name (the channel for the input) and an abstraction (i.e., a function from names to processes) are required. Similarly, restrictions are encoded using abstractions. This is the core of higher order syntax.

It is easy to define a translation function $\mathbf{t}$ from π-calculus processes, as defined in Section 2, to elements of type *proc*. Most of the π-calculus constructors are translated in the obvious way, but for input prefix and restrictions:

$$\mathbf{t}(x(y).p) = in_{pref}(x, \lambda y : name.\ \mathbf{t}(p)) \qquad \mathbf{t}((\nu y)p) = \nu(\lambda y : name.\ \mathbf{t}(p)).$$

Accordingly, π-calculus processes will be often pretty-printed, following the notation of Section 2.

The **judgments** we introduce are the following:

$$\begin{array}{lcl} \longmapsto & : & proc \to label \to proc \to Type \\ \longmapsto\!\!\!\!\!\!\twoheadrightarrow & : & proc \to (name \to label) \to (name \to proc) \to Type \end{array}$$

Table 2 LF rules

TAU	$: \Pi_{p:proc}.\ \tau.p \overset{\tau}{\longmapsto} p$
OUT	$: \Pi_{p:proc}\Pi_{x,y:name}.\ \bar{x}y.p \overset{\bar{x}y}{\longmapsto} p$
IN	$: \Pi_{p:name\to proc}\Pi_{x:name}.\ in_{pref}(x,p) \overset{\lambda z:name.xz}{\longmapsto\!\!\!\!\rightarrow} p$
SUM^l_f	$: \Pi_{p,q,r:proc}\Pi_{\alpha:label}.\ p \overset{\alpha}{\longmapsto} r \to (p{+}q) \overset{\alpha}{\longmapsto} r$
SUM^l_b	$: \Pi_{p,q:proc}\Pi_{r:name\to proc}\Pi_{\alpha:name\to label}.\ p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} r \to (p{+}q) \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} r$
PAR^l_f	$: \Pi_{p,q,r:proc}\Pi_{\alpha:label}.\ p \overset{\alpha}{\longmapsto} r \to (p\vert q) \overset{\alpha}{\longmapsto} (r\vert q)$
PAR^l_b	$: \Pi_{p,q:proc}\Pi_{r:name\to proc}\Pi_{\alpha:name\to label}.\ p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} r \to$ $(p\vert q) \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} (\lambda x : name.\ (rx\vert q))$
COM^l	$: \Pi_{p,q,r:proc}\Pi_{s:name\to proc}\Pi_{x,y:name}.\ p \overset{\bar{x}y}{\longmapsto} r \to$ $q \overset{\lambda z:name.xz}{\longmapsto\!\!\!\!\rightarrow} s \to (p\vert q) \overset{\tau}{\longmapsto} (r\vert sy)$
RES_f	$: \Pi_{p,q:name\to proc}\Pi_{\alpha:label}.(\Pi_{x:name}.px \overset{\alpha}{\longmapsto} qx) \to \nu(p) \overset{\alpha}{\longmapsto} \nu(q)$
RES_b	$: \Pi_{p:name\to proc}\Pi_{q:name\to name\to proc}\Pi_{\alpha:name\to label}.$ $(\Pi_{x:name}.\ px \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} \lambda z : name.\ qzx) \to$ $\nu(p) \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} \lambda z : name.\ \nu(qz)$
$OPEN$	$: \Pi_{p,q:name\to proc}\Pi_{x:name}.$ $(\Pi_{y:name}.\ py \overset{\bar{x}y}{\longmapsto} qy) \to \nu(p) \overset{\lambda z:name.\ \bar{x}z}{\longmapsto\!\!\!\!\rightarrow} q$
$CLOSE^l$	$: \Pi_{p,q:proc}\Pi_{r,s:name\to proc}\Pi_{x:name}.\ p \overset{\lambda z:name.\ \bar{x}z}{\longmapsto\!\!\!\!\rightarrow} r \to$ $q \overset{\lambda z:name.\ xz}{\longmapsto\!\!\!\!\rightarrow} s \to (p\vert q) \overset{\tau}{\longmapsto} \nu(\lambda z : name.\ (rz\vert sz))$
$BANG_f$	$: \Pi_{p,q:proc}\Pi_{\alpha:label}.\ p\vert!p \overset{\alpha}{\longmapsto} q \to\ !p \overset{\alpha}{\longmapsto} q$
$BANG_b$	$: \Pi_{p:proc}\Pi_{q:name\to proc}\Pi_{\alpha:name\to label}.\ p\vert!p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} q \to\ !p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} q$
$COND_f$	$: \Pi_{p,q:proc}\Pi_{\alpha:label}\Pi_{x:name}.\ p \overset{\alpha}{\longmapsto} q \to [x{=}x]p \overset{\alpha}{\longmapsto} q$
$COND_b$	$: \Pi_{p:proc}\Pi_{q:name\to proc}\Pi_{\alpha:name\to label}\Pi_{x:name}.$ $p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} q \to [x{=}x]p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} q$

Judgment $\longmapsto$ is used to encode the free transition assertion. Its meaning should be clear. We will often write $p \overset{\alpha}{\longmapsto} p'$ instead of $\longmapsto\ p\,\alpha\,p'$. Judgment $\longmapsto\!\!\!\!\rightarrow$ is used to encode bound transitions. We will write $p \overset{\alpha}{\longmapsto\!\!\!\!\rightarrow} p'$ instead of $\longmapsto\!\!\!\!\rightarrow (p,\alpha,p')$. In this case, both the label and the target are abstractions (of a *label* and of a *proc* respectively).

Rules for $\longmapsto$ and $\longmapsto\!\!\!\!\rightarrow$ appear in Table 2 (rules SUM^r_f, SUM^r_b, PAR^r_f, PAR^r_b, COM^r and $CLOSE^r$ have been omitted). Notice that some of the rules of Table 1 have two counterparts, according to whether they refer to free or bound transitions. Some comments on the rules of Table 2 are in order.

- No explicit axiomatization of α-conversion is needed. Since the π-calculus binders are explained in terms of abstractions, α-conversion is delegated to the metalanguage and hence it is for free.
- In rule IN, both the label and the target process are abstractions; the idea is that, when an input occurs, the target is applied to the effectively received name, so that this name replaces all the occurrences of the placeholder in the target process. This happens, for instance, in rule COM: in this case, the target s of the input transition is applied to the name y, which is the message of the output transition.
- Consider now rules RES: in Table 1 these rules have a side-condition; namely a transition of p with label α can be performed also by $(\nu x)p$ *provided* x does not appear among the names of α. This side-condition is expressed in the LF encoding by requiring that the process px perform the transition independently from the name x, which is chosen to instantiate abstraction p. In this way, the variable x, being bound, behaves as "generic" hence different from all the names in α.
- Rule $OPEN$ is similar; in this case, however, according to Table 1, the value of the free output has to coincide with the name of the restriction. The free output becomes a bound output, so the restriction is removed in the target state, which then becomes an abstraction.
- In rule $CLOSE$ an input and a bound output are synchronized. In this case, the fact that the values of the two transitions should be identified is obtained by combining the target processes $\lambda z : name.\ rz$ and $\lambda z : name.\ sz$ into $\lambda z : name.\ (rz|sz)$.

The operational semantics we have presented here has some advantages w.r.t. the ordinary one, deriving from the usage of functions to model bound transitions. First of all, one can eliminate those explicit side-conditions, which are needed in the ordinary semantics to enforce that the bound names be fresh, so as to avoid name clashing. In the LF presentation, the side conditions are implicit in the higher-order nature of the type encoding the rule. Bound names remain *bound* also when the transition is performed, since abstractions are used for the label and for the target process. As we will see, the names which have been abstracted in the targets of bound transitions will be instantiated in the definition of bisimulation. Another advantage is that name substitution is not required in the operational semantics (and in the definition of bisimulation): function applications are sufficient to represent all the required forms of name instantiation.

Let Σ_π denote the list of variable declarations introduced above. The signature Σ_π is the LF presentation of the π-calculus. Throughout this paper we shall assume that we have an infinite supply of different variables of type $name$, standing for the names in $\mathcal{N}$, and we shall work, even without explicit mention, in the environment Γ_π defined as follows: $\Gamma_\pi \cong \Sigma_\pi \cup \{x_i : name\}_{i \in \mathbf{N}}$. We shall denote with the name of the type (i.e. $name$, $label$, $proc$) the collection of all canonical (normal) well-typed terms of that type, derivable in the context Γ_π. Moreover, for $a \subseteq name$, we shall denote with $label_a$ the subset of $label$, consisting only of normal forms whose free variables appear in a.

In view of the remarks above, the set $fn(p)$ of free names of $p \in proc$ is therefore simply the set of *free variables* of type $name$, occurring in the canonical (normal) form of p.

It is possible to establish a precise correspondence between *LF* judgements derivable in the environment Γ_π and the corresponding concepts phrased in terms of the system in Section 2. This takes the form of an *Adequacy Theorem* for the encoding. Here, we do this just for transitions. The proof of the following proposition is by induction on the structure of derivations (only if part) and on the structure of *normal forms* (if part).

Proposition 3.1.1 (adequacy) *Let p, q be π-calculus processes and x, y be names in Γ_π; then:*

- $p \xrightarrow{\tau} q \iff$ *the type* $p \overset{\tau}{\longmapsto} q$ *is inhabited in* Γ_π.
- $p \xrightarrow{\bar{x}y} q \iff$ *the type* $p \overset{\bar{x}y}{\longmapsto} q$ *is inhabited in* Γ_π.
- $p \xrightarrow{x(y)} q \iff$ *the type* $p \overset{\lambda y:name.\ xy}{\longmapsto\!\!\!\!\!\rightarrow} \lambda y : name.\ q$ *is inhabited in* Γ_π.
- $p \xrightarrow{\bar{x}(y)} q \iff$ *the type* $p \overset{\lambda y:name.\ \bar{x}y}{\longmapsto\!\!\!\!\!\rightarrow} \lambda y : name.\ q$ *is inhabited in* Γ_π.

4 LATE BISIMULATION

Here we present the definition of late strong bisimulation making use of the *LF* presentation of the π-calculus, Σ_π. Using Proposition 3.1.1, one can show that it is equivalent to that in [12].

Definition 4.0.1 (late strong bisimulation) *A symmetric relation $\mathcal{R} \subseteq proc \times proc$ is a* late strong bisimulation *if and only if, $\forall p, q : proc$, whenever $p\,\mathcal{R}\,q$:*

- $\forall \alpha : label.\ \forall r : proc.\ p \overset{\alpha}{\longmapsto} r \to (\exists s : proc.\ q \overset{\alpha}{\longmapsto} s \wedge r\,\mathcal{R}\,s)$,
- $\forall x : name.\ \forall r : name \to proc.\ p \overset{\lambda z:name.\bar{x}z}{\longmapsto\!\!\!\!\!\rightarrow} r \to$
  $(\exists s : name \to proc.\ q \overset{\lambda z:name.\bar{x}z}{\longmapsto\!\!\!\!\!\rightarrow} s \wedge \forall y : name.\ (y \notin fn(r), fn(s) \to$
  $ry\,\mathcal{R}\,sy))$, *and*
- $\forall x : name.\ \forall r : name \to proc.\ p \overset{\lambda z:name.xz}{\longmapsto\!\!\!\!\!\rightarrow} r \to$
  $(\exists s : name \to proc.\ q \overset{\lambda z:name.xz}{\longmapsto\!\!\!\!\!\rightarrow} s \wedge \forall y : name.\ ry\,\mathcal{R}\,sy)$.

Let $p, q : proc$; p is late strongly bisimilar *to q if there exists a late strong bisimulation $\mathcal{R} \subseteq proc \times proc$ such that $p\,\mathcal{R}\,q$. The union of all late strong bisimulations (denoted by $\sim$) is a late strong bisimulation itself, called* late strong bisimilarity.

It is well known that late strong bisimularity is not a congruence with respect to *input prefix*. The largest congruence included in late bisimilarity can be obtained by allowing to instantiate the names in the processes to be compared, before starting to match their transitions.

Definition 4.0.2 (late strong congruence) *Two processes $p, q \in proc$ are* late strong congruent*, written $p \sim^c q$, if $p\sigma \sim q\sigma$ for all substitutions σ.*

Now we give the LF representation of *late weak bisimilarity*. First, we encode the notion of weak transition, by extending the LF signature Σ_π with seven new variable declarations, corresponding to two new judgments and five new rules:

$$
\begin{array}{ll}
\Longmapsto & : \ proc \to label \to proc \to Type \\
\Longmapsto\!\!\!\!\!\!\to & : \ proc \to (name \to label) \to (name \to proc) \to Type \\
weak1_\tau & : \Pi_{p:proc}.\ p \overset{\tau}{\Longmapsto} p \\
weak2_\tau & : \Pi_{p,q,r:proc}.\ p \overset{\tau}{\Longmapsto} q \to q \overset{\tau}{\longmapsto} r \to p \overset{\tau}{\Longmapsto} r \\
weak_{out} & : \Pi_{p,q,r,s:proc}\Pi_{x,y:name}.\ p \overset{\tau}{\Longmapsto} q \to q \overset{\bar{x}y}{\longmapsto} r \to \\
& \quad r \overset{\tau}{\Longmapsto} s \to p \overset{\bar{x}y}{\Longmapsto} s \\
weak_{in} & : \Pi_{p,q:proc}\Pi_{r:name\to proc}\Pi_{x:name}.\ p \overset{\tau}{\Longmapsto} q \to \\
& \quad q \overset{\lambda z:name.xz}{\longmapsto\!\!\!\!\!\to} r \to p \overset{\lambda z:name.xz}{\Longmapsto\!\!\!\!\!\!\to} r \\
weak_{bout} & : \Pi_{p,q:proc}\Pi_{r,s:name\to proc}\Pi_{x:name}.\ p \overset{\tau}{\Longmapsto} q \to \\
& \quad q \overset{\lambda z:name.\bar{x}z}{\longmapsto\!\!\!\!\!\to} r \to (\Pi_{z:name}.rz \overset{\tau}{\Longmapsto} sz) \to p \overset{\lambda z:name.\bar{x}z}{\Longmapsto\!\!\!\!\!\!\to} s
\end{array}
$$

Notice that, in the case of weak input transitions, no τ-actions are allowed after the visible transitions. This is standard for the weak semantics of late π-calculus.

Definition 4.0.3 (late weak bisimulation) *A symmetric relation $\mathcal{R} \subseteq proc \times proc$ is a* late weak bisimulation *if and only if, whenever $p\,\mathcal{R}\,q$ then:*

- $\forall \alpha : label.\ \forall r : proc.\ p \overset{\alpha}{\Longmapsto} r \to (\exists s : proc.\ q \overset{\alpha}{\Longmapsto} s \land r\ \mathcal{R}\ s)$,
- $\forall x : name.\ \forall r : name \to proc.\ p \overset{\lambda z:name.\bar{x}z}{\Longmapsto\!\!\!\!\!\!\to} r \to$
  $(\exists s : name \to proc.\ q \overset{\lambda z:name.\bar{x}z}{\Longmapsto\!\!\!\!\!\!\to} s \land \forall y : name.\ (y \notin fn(r), fn(s)) \to$
  $ry\ \mathcal{R}\ sy))$ *, and*
- $\forall x : name.\ \forall r : name \to proc.\ p \overset{\lambda z:name.xz}{\Longmapsto\!\!\!\!\!\!\to} r \to$
  $(\exists s : name \to proc.\ q \overset{\lambda z:name.xz}{\Longmapsto\!\!\!\!\!\!\to} s \land \forall y : name\ ry\ \mathcal{R}\ sy)$.

Let $p, q : proc$. Then p is late weak bisimilar *to q if there exists a late weak bisimulation $\mathcal{R} \subseteq proc \times proc$ such that $p\ \mathcal{R}\ q$. The union of all late weak bisimulations (denoted by $\approx$) is a late weak bisimulation itself, called* late weak bisimilarity.

Sometimes an alternative characterization of weak bisimulation is used, in which strong transitions are used for the first process. The two definitions are equivalent. We give the version with weak transitions for both processes, since it can be captured more easily in the final approach.

5 FINAL SEMANTICS

In this section we show how to re-obtain by finality the notions of strong and weak bisimilarity defined in the previous section.

5.1 The final approach

We work in $Class^*$, the category whose objects are the classes of the Universe of $ZF_0^-(U)X_1$ and whose arrows are the functional classes. The theory $ZF_0^-(U)X_1$ is a theory of hypersets with atoms, i.e. a Zermelo-Frænkel-like set theory where the Axiom of Extensionality has been weakened so as to allow for a class U of atoms, and where the axiom of Foundation has been replaced by the *Antifoundation Axiom* X_1 of [6] (also known as AFA [1]). As was originally shown by Aczel ([1]), a large class of endofunctors $F : Class^* \to Class^*$ admit *final* F*-coalgebras* of the form $(\bar{X}_F, \iota_{\bar{X}_F})$, where $\iota_{\bar{X}_F}$ is the *identity*. Recall that an F-coalgebra morphism $h : (A, f) \to (B, g)$ amounts to a function $h : A \to B$, such that $g \circ h = F(h) \circ f$.

According to the final semantics approach (see [1, 2, 15, 9]), the operational semantics of a language is represented as an F*-coalgebra* for a suitable functor F, i.e. as a pair (A, f) such that $f : A \to F(A)$. If the functor is "well-behaved", then there exists a *final* F*-coalgebra* in $Class^*$, i.e. $(\bar{X}_F, \iota_{\bar{X}_F})$. Moreover, the unique arrow $\mathcal{M}_F : (A, f) \to (\bar{X}_F, \iota_{\bar{X}_F})$, induces an equivalence on A which can be characterized as the union of all F*-bisimulations* on (A, f). An F-bisimulation on (A, f) is the categorical counterpart of the ordinary notion of bisimulation. It is a relation $R \subseteq A \times A$ for which there exists an arrow $\gamma : R \to F(R)$, such that the projections $\pi_1,\ \pi_2$ can be lifted to F-coalgebra morphisms from (R, γ) to (A, f).

In the case of CCS-like languages, the appropriate functor is $F_{ord}(X) = \mathcal{P}(Label \times X)$. Then the syntactical F-coalgebra which embodies the operational semantics is $(Proc, f_{ord})$, where: $f_{ord}(p) = \{(\alpha, q) \mid p \xrightarrow{\alpha} q\}$. The equivalence captured by the unique morphism is the CCS bisimilarity corresponding to the operational semantics given by $\xrightarrow{\alpha}$. We will see that the functor (and the syntactical coalgebra) are more complex in the case of π-calculus.

5.2 Final description of late strong bisimilarity

In defining the functor F, and the syntactical F-coalgebra, for the strong late semantics of the π-calculus, we cannot re-use directly the standard functor F_{ord} described above. In fact, in the definition of late bisimulation, when two bound outputs are compared, we require that the names which are used to instantiate the placeholder in the pair of target processes are fresh for both processes (conditions $y \notin fn(r), fn(s)$ in Definitions 4.0.1 and 4.0.3). This is unproblematic when dealing with pairs of processes. In the co-algebraic approach, however, the arrow f, corresponding to the syntactical coalgebra, should describe the transitions of each process independently from its possible partners. One cannot use simply, as names for the bound outputs of p, the names which do not appear in p, otherwise equivalent processes with different sets of free names would give rise to different sets of transitions, and hence would receive different final semantics.

The solution we propose consists in associating to processes sets of names (of possible partners). We work with pairs of the form (a, p), where p is a process and $a \in \mathcal{P}_{fin}(name)$ is an estimate of the finite set of free names currently available for interaction: i.e., $fn(p) \subseteq a$. When two processes p and q are compared, it is sufficient to choose $a \in \mathcal{P}_{fin}(name)$ so that $fn(p), fn(q) \subseteq a$ and to compare (a, p) and (a, q). This idea of associating the set of free names to a π-calculus process has been used also in [4] to obtain a characterization of π-calculus based on ordinary transition systems and bisimulations.

In the sequel we fix a function $e : \mathcal{P}_{fin}(name) \to name$ such that, for all $a \subseteq name$, the value x_a of e on a is not in a. We shall denote by a^+ the finite set of names $a \cup \{x_a\}$.

Throughout this paper, we shall introduce functors and refer to their final coalgebras, without explicit proofs that they are indeed functors having final coalgebras. Using standard facts and the theorems in [15], one can always see immediately that this is the case.

Definition 5.2.1 *Let $F : Class^* \to Class^*$ be the functor:*

$$F(X) = \bigcup_{a \in \mathcal{P}_{fin}(name)} \{a\} \times (\mathcal{P}_{fin}((label_{a^+} \times X) \cup (label_{a^+} \times X)^{a^+}) \cup \{\perp\}).$$

The definition of F on arrows is canonical.

Proposition 5.2.1 *The functor F has a final coalgebra: $(\bar{X}_F, \iota_{\bar{X}_F})$.*

Definition 5.2.2 *Let $(\mathcal{P}_{fin}(name) \times proc, f)$ be the F-coalgebra defined as follows:*

$$f(a,p) = \begin{cases} (a,\ \{(\alpha, (a,q)) \mid p \overset{\alpha}{\longmapsto} q\} \cup \\ \quad \{(\bar{y}x_a, (a^+, qx_a)) \mid p \overset{\lambda z:name.\ \bar{y}z}{\longmapsto\!\!\!\!\twoheadrightarrow} q\} \cup \\ \quad \{\lambda x \in a^+. \\ \qquad \mathit{if}\ x \in a\ \mathit{then}\ (yx, (a, qx)) \\ \qquad \mathit{else}\ (yx_a, (a^+, qx_a) \mid p \overset{\lambda z:name.\ yz}{\longmapsto\!\!\!\!\twoheadrightarrow} q\}) & \mathit{if}\ fn(p) \subseteq a \\ (a, \perp) & \mathit{if}\ fn(p) \not\subseteq a \end{cases}$$

According to the definition of the functor F, the function f associates to each pair $(a, p) \in \mathcal{P}_{fin}(name) \times proc$ a pair (a, U). The component U is $\perp$, whenever a is not a correct estimate of the free names of p. Otherwise, $U \subseteq (label_{a^+} \times X) \cup (label_{a^+} \times X)^{a^+}$ represents the transitions that the process

p can perform. In the case of τ-actions, output, and bounded output, U is a subset of $\mathcal{P}_{fin}(name) \times proc$. In the case of input transitions, U is a function defined on a suitable finite set of names. This is arranged in such a way that all the names in a and the fresh name x_a are used as possible input values. In the case of bound output transitions, just name x_a is used. A single name, namely x_a, is chosen when a fresh name is required in a bound transition. This is not restrictive, since a single fresh name is sufficient for checking bisimulation equivalence. In this way, moreover, the function f maps each process (a, p) to a finite set U.

The following theorem is the main result of the paper: two π-calculus processes are equivalent according to Definition 4.0.1 if and only if they are equivalent according to the co-algebraic approach.

Theorem 5.2.1 *Let $p, q \in proc$ be such that $fn(p), fn(q) \subseteq a$. Then $p \sim q$ if and only if there exists an F-bisimulation, $\mathcal{R}$, on the coalgebra $(\mathcal{P}_{fin}(name) \times proc, f)$ such that $(a,p)\mathcal{R}(a,q)$.*

The proof of this theorem is based on the following lemma, which characterizes F-bisimulations. Notice that any *categorical bisimulation*, such as the F-bisimulation above, is symmetric by definition.

Lemma 5.2.1 *An F-bisimulation on the F-coalgebra $(\mathcal{P}_{fin}(name) \times proc, f)$ is a symmetric relation $\mathcal{R} \subseteq (\mathcal{P}_{fin}(name) \times proc) \times (\mathcal{P}_{fin}(name) \times proc)$, such that $(a,p)\mathcal{R}(a',q)$ implies $a = a'$ and one of the two conditions:*

- $fn(p) \not\subseteq a \ \wedge \ fn(q) \not\subseteq a,$
- $fn(p) \subseteq a \ \wedge \ fn(q) \subseteq a \ \wedge$
  $[\forall \alpha : label.\ \forall r : proc.\ p \stackrel{\alpha}{\longmapsto} r \to (\exists s : proc.\ q \stackrel{\alpha}{\longmapsto} s \wedge (a,r)\mathcal{R}(a,s))] \ \wedge$
  $[\forall x : name.\ \forall r : name \to proc.\ p \stackrel{\lambda z:name.xz}{\longmapsto\!\!\!\!\rightarrow} r \to$
  $(\exists s : name \to proc.\ q \stackrel{\lambda z:name.xz}{\longmapsto\!\!\!\!\rightarrow} s \wedge\ \forall x \in a.\ (a, rx)\mathcal{R}(a, sx))] \ \wedge$
  $[\forall \alpha : name \to label.\ \forall r : name \to proc.\ p \stackrel{\alpha}{\longmapsto\!\!\!\!\rightarrow} r \to$
  $(\exists s : name \to proc.\ q \stackrel{\alpha}{\longmapsto\!\!\!\!\rightarrow} s \wedge\ (a^+, rx_a)\mathcal{R}(a^+, sx_a))].$

The following proposition characterizes F-bisimilarity by finality, and, by Theorem 5.2.1, also bisimilarity on π-calculus processes.

Proposition 5.2.2 *The equivalence induced by the unique morphism $\mathcal{M}_F : (\mathcal{P}_{fin}(name) \times proc, f) \to (\bar{X}_F, \iota_{\bar{X}_F})$ coincides with the union of all F-bisimulations, $\sim_F^{(\mathcal{P}_{fin}(name)\times proc, f)}$, on the F-coalgebra $(\mathcal{P}_{fin}(name) \times proc, f)$.*

5.3 Alternative semantics

The functor F of Definition 5.2.1 is by no means the unique possible. Many other kinds of functors can be used just as well to provide a final account of strong bisimilarity. We chose such an F because we feel that it is simple and yet perspicuous.

By way of example we shall present an interesting alternative, which eliminates, at the price of an increase in complexity, the somewhat unpleasant fact that F yields only indirectly the semantics of a process. We use the notation of definitions 5.2.1 and 5.2.2.

Definition 5.3.1 *Let $G : Class^* \to Class^*$ be the functor:*

$G(X) = [F(\mathcal{P}_{fin}(name) \times X)^{\mathcal{P}_{fin}(name)}]_{/\simeq}$

where $[F(\mathcal{P}_{fin}(name) \times X)^{\mathcal{P}_{fin}(name)}]_{/\simeq}$ denotes the set of equivalence classes of elements of $F(\mathcal{P}_{fin}(name) \times X)^{\mathcal{P}_{fin}(name)}$ modulo the equivalence:

$g \simeq g' \Longleftrightarrow \exists a \in \mathcal{P}_{fin}(name).\ \forall b \in \mathcal{P}_{fin}(name).\ a \subseteq b \Longrightarrow g(b) = g'(b).$

The definition of G on arrows is canonical.

Definition 5.3.2 *Let $(proc, g)$ be the G-coalgebra defined as follows:*

$g(p) = [\lambda a \in \mathcal{P}_{fin}(name).(a, f(a, p))]_{\simeq}.$

Using standard arguments, one can show that the functor G is well-behaved and that the following proposition holds.

Proposition 5.3.1 *Let $p, q \in proc$. Then*

$\mathcal{M}_G(p) = \mathcal{M}_G(q) \Longleftrightarrow \exists a \in \mathcal{P}_{fin}(name).\ \mathcal{M}_F((a, p)) = \mathcal{M}_F((a, q)).$

5.4 Final description of late strong congruence

The difficulty in capturing by finality late strong congruence (see Definition 4.0.2) lies in the fact that substitutions of names are applied only to the starting pair of processes. But according to the final semantics paradigm, any functor has to act uniformly on all the processes; and so there is no natural notion of "initial pair". At the price of a little awkwardness, however, we can proceed as follows. We split the type of names into two infinite subsets, e.g $name_{even}$ consisting of those names with an *even* index and $name_{odd}$, consisting of those names with an *odd* index. We assume that initial π-calculus processes have only names in $name_{odd}$. By considering processes only "up to" substitutions of the form $\sigma : name \to name_{even}$, which are *injective* on $name_{even}$, we can then sidestep the problem given by the absence of an "initial pair". An adequate, uniform definition of a functor F_{lc} can then be defined, whose largest F_{lc}-bisimulation can be shown to be the required congruence.

More specifically, we proceed as follows. First of all we assume that for all $a \in \mathcal{P}_{fin}(name)$, $x_a \in name_{even}$. Let:

$\mathcal{S} = \{\sigma : name \to name_{even} \mid \sigma \text{ injective on } name_{even}\}.$

Definition 5.4.1 *Let $F_{lc} : Class^* \to Class^*$ be the functor:*

$$F_{lc}(X) = \{\perp\} \cup (\bigcup_{a \in \mathcal{P}_{fin}(name)} \{a\} \times \mathcal{P}_{fin}((label_{a^+} \times X) \cup (label_{a^+} \times X)^{a^+}))^{\mathcal{S}}.$$

The definition of F_{lc} on arrows is canonical.

The following proposition is straightforward.

Proposition 5.4.1 *The functor F_{lc} has a final coalgebra $(\bar{X}_{F_{lc}}, \iota_{\bar{X}_{F_{lc}}})$.*

Definition 5.4.2 *Let $(\mathcal{P}_{fin}(name) \times proc, f_{lc})$ be the F_{lc}-coalgebra:*

$$f_{lc}(a,p) = \begin{cases} \perp & \text{if } fn(p) \not\subseteq a \\ \lambda\sigma \in \mathcal{S}.\ f(\sigma^*(a), p\sigma) & \text{if } fn(p) \subseteq a \end{cases}$$

where $\sigma^*(a) = \{\sigma(x) \mid x \in a\}$.

Finally we have the following proposition, whose proof is standard.

Proposition 5.4.2 *Let $p, q \in proc$ such that $fn(p), fn(q) \subseteq (a \cap name_{odd})$. Then $p \sim^c q$ if and only if $\mathcal{M}_{F_{lc}}((a,p)) = \mathcal{M}_{F_{lc}}((a,q))$.*

5.5 Final description of late weak bisimilarity

Weak bisimulations differ from strong bisimulations only in that *weak transitions* are used instead of strong transitions. Contrary to what happens in other semantical settings based on the notion of *continuity* (e.g. domain theory or complete metric spaces), this is not at all problematic in our purely set-theoretical setting. We can easily obtain also in the weak context all the results we have for strong bisimulation. We only need to modify slightly the functor F of Definition 5.2.1 and the coalgebra of Definition 5.2.2, taking into account the fact that weak processes can be infinitely branching.

Definition 5.5.1 *Let $F_w : Class^* \to Class^*$ be the functor:*

$$F_w(X) = \bigcup_{a \in \mathcal{P}_{fin}(name)} \{a\} \times (\mathcal{P}_{<2^{\aleph_0}}((label_{a^+} \times X) \cup (label_{a^+} \times X)^{a^+}) \cup \{\perp\}).$$

The definition of F_w on arrows is canonical.

Definition 5.5.2 *Let $(\mathcal{P}_{fin}(name) \times proc, f_w)$ be the F_w-coalgebra:*

$$f_w(a,p) = \begin{cases} (a, \{(\alpha, (a,q)) \mid p \overset{\alpha}{\Longmapsto} q\} \cup & \\ \quad \{(\bar{y}x_a, (a^+, qx_a)) \mid p \overset{\lambda z:name.\ \bar{y}z}{\Longmapsto} q\} \cup & \\ \quad \{\lambda x \in a^+. & \\ \qquad \text{if } x \in a \text{ then } (yx, (a, qx)) & \\ \qquad \text{else } (yx_a, (a^+, qx_a) \mid p \overset{\lambda z:name.\ yz}{\Longmapsto} q\}) & \text{if } fn(p) \subseteq a \\ (a, \perp) & \text{if } fn(p) \not\subseteq a \end{cases}$$

Theorem 5.5.1 *Let $p, q \in proc$ be such that $fn(p), fn(q) \subseteq a$. Then $p \approx q$ if and only if there exists an F_w-bisimulation on the F_w-coalgebra $(\mathcal{P}_{fin}(name) \times proc, f_w)$ such that $(a,p)\mathcal{R}(a,q)$.*

Proposition 5.5.1 *The equivalence induced by the unique morphism $\mathcal{M}^w_{F_w}$: $(\mathcal{P}_{fin}(name)\times proc, f_w) \to (\bar{X}_{F_w}, \iota_{\bar{X}_{F_w}})$ coincides with the union of all F_w-bi-simulations, $\sim_{F_w}^{(\mathcal{P}_{fin}(name)\times proc,f_w)}$, on the coalgebra $(\mathcal{P}_{fin}(name)\times proc, f_w)$.*

6 APPLICATIONS

As an application of the final semantics apparatus, that we have erected so far, we give a semantical proof of the law $p|q \sim q|p$. Of course this proof ultimately corresponds to the usual proof by coinduction, but the machinery of final semantics allows to express it in a more structured way. In our setting this amounts to showing that $\forall a \in \mathcal{P}_{fin}(name).\ \mathcal{M}_F(\langle a, p|q\rangle) = \mathcal{M}_F(\langle a, q|p\rangle)$. To this end, because of the particular nature of the π-calculus, we will have to show more in general that

$$\begin{array}{ll}\forall a \in \mathcal{P}_{fin}(name). & \mathcal{M}_F(\langle a, \nu(\lambda z_1.(\cdots \nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots))\rangle) = \\ & \mathcal{M}_F(\langle a, \nu(\lambda z_1.(\cdots \nu(\lambda z_n.\ qz_1\cdots z_n|pz_1\cdots z_n)\cdots))\rangle)\end{array}$$

holds for all naturals $n \in \mathbf{N}$ and $p, q \in name^n \to proc$, where $name^0 \to proc = proc$ and $name^{n+1} \to proc = (name^n \to proc)^{name}$. In the case $n = 0$ this reduces to $\forall a \in \mathcal{P}_{fin}(name).\ \mathcal{M}_F(\langle a, p|q\rangle) = \mathcal{M}_F(\langle a, q|p\rangle)$.

We will achieve our goal exploiting the finality of (X_F, ι_F) among F-coalgebras. First we introduce a suitable F-coalgebra (B, β), and two functions $g_1, g_2 : B \to \mathcal{P}_{fin}(name)\times proc$, such that for suitable elements $u_{a,n,p,q}$ in B, $g_1(u_{a,n,p,q}) = \langle a, \nu(\lambda z_1.(\cdots \nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots))\rangle$ while $g_2(u_{a,n,p,q}) = \langle a, \nu(\lambda z_1.(\cdots \nu(\lambda z_n.\ qz_1\cdots z_n|pz_1\cdots z_n)\cdots))\rangle$.

We show then that both $\mathcal{M}_F\circ g_1$ and $\mathcal{M}_F \circ g_2$ are F-coalgebra morphisms from (B, β) to the final F-coalgebra (X_F, ι_F). Therefore they coincide by finality.

Here are the details. Let:

$$\begin{array}{l}B = \mathcal{P}_{fin}(name)\times (\bigcup_{n\in\mathbf{N}}\{n\}\times(name^n\to proc)\times(name^n \to proc));\\ g_1(\langle a, \langle n,p,q\rangle\rangle) = \langle a, \nu(\lambda z_1(\cdots \nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots))\rangle;\\ g_2(\langle a, \langle n,p,q\rangle\rangle) = \langle a, \nu(\lambda z_1(\cdots \nu(\lambda z_n.\ qz_1\cdots z_n|pz_1\cdots z_n)\cdots))\rangle.\end{array}$$

The definition of β is rather involved, since it has to take into account the different kinds of transitions:

$$\beta(\langle a, \langle n,p,q\rangle\rangle) = \left\{\begin{array}{ll}\langle a, \bot\rangle & \text{if } fn(p), fn(q) \not\subseteq a\\ \langle a, A_{par}\cup A_{open}\cup A_{close}\rangle & \text{if } fn(p), fn(q)\subseteq a\end{array}\right.$$

where:

$$\begin{array}{rcl}A_{par} & = & \{\langle \alpha, \langle a, \langle n, r, s\rangle\rangle\rangle \mid \\ & & \quad \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots)) \stackrel{\alpha}{\longmapsto} \\ & & \quad \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ rz_1\cdots z_n|sz_1\cdots z_n)\cdots))\}\\ & \cup & \{\langle \bar{y}x_a, \langle a^+, \langle n, rx_a, sx_a\rangle\rangle\rangle \mid\end{array}$$

$$
\begin{array}{rcl}
 & & \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots)) \overset{\lambda x.\bar{y}x}{\longmapsto\!\!\!\!\twoheadrightarrow} \\
 & & \lambda x.\nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ rxz_1\cdots z_n|sxz_1\cdots z_n)\cdots))\} \\
 & \cup & \{\lambda x\in a^+.\ \text{ if } x\in a \text{ then } \langle yx,\langle a,\langle n,rx,sx\rangle\rangle\rangle \\
 & & \text{else } \langle yx_a,\langle a^+,\langle n,rx_a,sx_a\rangle\rangle\rangle \mid \\
 & & \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots)) \overset{\lambda x.yx}{\longmapsto\!\!\!\!\twoheadrightarrow} \\
 & & \lambda x.\nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ rxz_1\cdots z_n|sxz_1\cdots z_n)\cdots))\}, \\
A_{open} & = & \{\langle\bar{y}x_a,\langle a^+,\langle n-1,rx_a,sx_a\rangle\rangle\rangle \mid \\
 & & \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots)) \overset{\lambda x:name.\ \bar{y}x}{\longmapsto\!\!\!\!\twoheadrightarrow} \\
 & & \quad\lambda x.\nu(\lambda z_1.(\cdots\nu(\lambda z_{n-1}.\ rxz_1\cdots z_{n-1}|sxz_1\cdots z_{n-1})\cdots))\}, \\
A_{close} & = & \{\langle\tau,\langle a,\langle n+1,r,s\rangle\rangle\rangle \mid \\
 & & \nu(\lambda z_1.(\cdots\nu(\lambda z_n.\ pz_1\cdots z_n|qz_1\cdots z_n)\cdots)) \overset{\tau}{\longmapsto} \\
 & & \quad\nu(\lambda z_1.(\cdots\nu(\lambda z_n.\nu(\lambda x.\ rz_1\cdots z_nx|sz_1\cdots z_nx))\cdots))\}.
\end{array}
$$

It is interesting to point out that the categorical methodology used above, i.e. defining suitable coalgebras and exploiting uniqueness of the final morphism, can be uniformly applied to other coinductive proofs of properties of processes. This methodology is also quite general. For instance it could be used to define coinductively a semantical operator of parallel composition.

7 CONCLUDING REMARKS

In this paper we have given a higher order presentation of the π-calculus and its operational semantics, using LF as metalanguage. This has allowed us to do away with α-conversion, and any explicit mentioning in rules of side-conditions enforcing freshness of names. Building on this operational semantics, we have equipped the π-calculus with a final semantics in both the strong and the weak context. We have presented only late bisimulation, since it is easy to extend the above results to *early* [13] and to *open* [16] bisimulations, see [10] for more details.

One can reasonably question as to how crucial the LF encoding is for the purposes of the paper. Probably, more investigation is needed to give a definitive answer. In principle, being a general metalanguage, which usually provides adequate representations of object languages, LF could be explained away. We feel however, that in this context, LF helps in separating concerns and it greatly simplifies the treatment of binding operators and bound variables.

We did not deal with the mismatch operator, $[\ \neq\ ]$. We could have easily done so at the price of an increase in the complexity of the LF encoding of the π-calculus. In fact, in order to produce an adequate encoding of the mismatch operator, it is not enough to add an explicit judgement to express *name diversity* and the rule for mismatch. Because of technical reasons, we need to modify also all rules in Table 2 which have a premise with a locally bound name. The fact that such a name is different from all the ones which can possibly occur in the global parameters has to be explicitly asserted. Perhaps this increase in complexity is an indication that such an operator has not as an immediate semantics as the others.

Rather problematic was the extension of the theory to obtain final characterizations of the various bisimulation congruences. In fact, neither strong nor weak late bisimulations are congruences. We have presented a possible solution for dealing with late strong congruence. In the case of weak late congruence, besides the problem of instantiating the names in the initial pair of processes, the usual problem due to τ-actions and summation also occurs. This second issue can be easily solved since we are working in a hyperset setting, using the technique proposed in [2] for CCS.

We have utilized the category $Class^*$ throughout our paper for uniformity with [2, 15]; but clearly much smaller categories are sufficient to carry out our constructions. For instance, we could have used the category of *sets of hereditarily finite hypersets*, in dealing with strong late bisimilarity, and the category of *sets of hereditarily countable hypersets*, in dealing with weak late bisimilarity. As a side remark related to the latter case, let us point out again, that it is the low mathematical overhead of our approach, that allows us to deal smoothly with arbitrary countably branching weak processes. Contrary to what happens in mathematically more constrained semantical settings, we do not need to worry about closedness or compactness of denotations.

A more thorough comparison of the final semantics for π-calculus presented here and the initial semantics presented in [5, 17] is called for, especially with respect to the metalanguage used in representing the language.

REFERENCES

[1] P.Aczel. Non-wellfounded sets. Number 14, Lecture Notes CSLI, 1988.

[2] P.Aczel. Final universes of processes. LNCS 802. Springer, 1993.

[3] C.Cornes *et al.* The Coq proof assistant reference manual. Version 5.10. INRIA, Rocquencourt, 1995.

[4] G.-L.Ferrari, U.Montanari, P.Quaglia. The weak late π-calculus semantics as observation equivalence. LNCS 962. Springer, 1995.

[5] M.Fiore, E.Moggi, D.Sangiorgi. A fully-abstract model for the π-calculus. In *LICS'96*, IEEE, 1996.

[6] M.Forti and F.Honsell. Set theory with free construction principles. Annali Scuola Normale Sup. Pisa, Cl. Sci., (IV), 10:493–522, 1983.

[7] R.Harper, F.Honsell, G.Plotkin. A framework for defining logics. *JACM*, 40(1):143–184, 1993.

[8] F.Honsell and M.Lenisa. Final semantics for untyped λ-calculus. In *TLCA'95*, LNCS 902, Springer, 1995.

[9] M.Lenisa. Final semantics for a higher order concurrent language. In *CAAP'96*, LNCS 1059, Springer, 1996.

[10] M.Lenisa. *Themes in final semantics.* PhD Thesis, Pisa, 1996.
[11] R.Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, NATO ASI Series F, Vol. 94. Springer, 1993.
[12] R.Milner, J.Parrow, D.Walker. A calculus of mobile processes. *Inf. and Comp.*, 100:1–77, 1992.
[13] R.Milner, J.Parrow, D.Walker. Modal logic for mobile processes. *Th. Comp. Sci.*, 114:149–171, 1993.
[14] J.J.M.M.Rutten. Universal coalgebra: a theory of systems. CWI Techical Report CS-R9652, 1996.
[15] J.J.M.M.Rutten and D.Turi. On the foundations of final semantics. In *REX'92*, LNCS 666. Springer, 1993.
[16] D.Sangiorgi. A theory of bisimulation for the π-calculus. In *CONCUR'93*, LNCS 715. Springer, 1993.
[17] I.Stark. A fully abstract domain model for the π-calculus. In *Proc. LICS'96*, IEEE Press, 1996.

BIOGRAPHY

Furio Honsell is Full Professor in Foundations of Computer Science at the University of Udine (Italy) since 1990. He has received a degree in Mathematics from the Scuola Normale Superiore in Pisa (Italy) in 1980. He has held research positions at the Computer Science Departments of the Universities of Torino and Edinburgh. His main areas of interest are in the Foundations of Mathematics and Computer Science as well as in Computer Assisted Formal Reasoning.

Marina Lenisa has received her PhD in Computer Science in 1998 at the Pisa-Udine Universities Consortium. Her main areas of interest are in Semantics and Formal Methods in Computer Science.

Ugo Montanari is Full Professor in Foundation of Computer Science at the University of Pisa (Italy) since 1976. He received a degree in Electronic Engineering from the Politecnico di Milano (Italy) in 1966. He took sabbatical leaves of absence in 1987 and 1997 at SRI Int., Menlo Park (USA). His main areas of interest are Semantics of Concurrency, Constraint Programming and Formal Methods.

Marco Pistore is currently doing his PhD in Computer Science at the University of Pisa (Italy). His main areas of interest are in Theory of Concurrency and in Formal Methods in Verification.