Modeling Fresh Names in the π-calculus Using Abstractions*

Roberto Bruni[1], Furio Honsell[2], Marina Lenisa[2], Marino Miculan[2]

[1] Dipartimento di Informatica, Università di Pisa,
Via F. Buonarroti 2, 56127 Pisa, ITALY.
e-mail: bruni@di.unipi.it
[2] Dipartimento di Matematica e Informatica, Università di Udine,
Via delle Scienze 206, 33100 Udine, ITALY.
e-mail: honsell,lenisa,miculan@dimi.uniud.it

Abstract. In this paper, we model fresh names in the π-calculus using *abstractions* w.r.t. a new binding operator θ. Both the theory and the metatheory of the π-calculus benefit from this simple extension. The operational semantics of this new calculus is *finitely branching*. Bisimulation can be given without mentioning any constraint on names, thus allowing for a straightforward definition of a *coalgebraic* semantics. This is cast within a category of coalgebras over algebras with infinitely many unary operators, in order to capitalize on θ. Following previous work by Montanari and Pistore, we present also a *finite* representation for *finitary* processes and a finite state verification procedure for bisimilarity, based on the new notion of θ-*automaton*. Finally, we improve previous encodings of the π-calculus in the Calculus of Inductive Constructions.

Introduction

The π-calculus [15, 21] is a process calculus which provides a conceptual framework for understanding mobility via name passing. Processes can communicate in a network whose topology can change dynamically by passing, possibly local, channel names. As for any other foundational calculus, we need strong mathematical tools for expressing mobile systems and reasoning about their behaviours. However, due to the peculiar behaviour of mobile processes and names, the well-known tools and techniques which have been developed successfully for CCS-like languages cannot be straightforwardly extended to the π-calculus.

At the syntactic level, we have the problematic issue of binders and scope of local names. At the operational semantic level, we have the issue of ensuring freshness conditions for names. At the model-theoretic level, we have the issue of providing a coalgebraic (final) semantics. Finally, from the practical point of view, a finite representation for finitary processes is desirable [10, 13, 16].

* Research supported by the MIUR Project COFIN 2001013518 COMETA, by the UE projects IST-2000-29001 TYPES, IST-2001-33477 DART and IST-2001-32747 AGILE. The first author is also partially supported by an Italian CNR fellowship for research on Information Sciences and Technologies and by the CS Department of the University of Illinois at Urbana-Champaign.

All the above issues have been considered in many previous papers. Often, reformulations of the same calculus are introduced, in order to cope with these problematic issues. The question is *how* to present "best" the calculus, in order to achieve the mathematical structure we need for reasoning on its core computational aspects. The answer to this question depends on the metalogical formalism in which we define the calculus. Recently, for the π-calculus many reformulations, in different metalogics, have been presented. A first-order, de Bruijn-like approach is adopted in [4,16], where processes can be equipped with explicit *permutations* of names. A second-order approach, based on Higher-Order Abstract Syntax, is adopted in [10,11,14], where most issues about freshness of names are simplified, taking advantage of the metalogic notion of capture-avoiding substitution. A somehow mid-way approach is in [7], where the reformulation is given in the logic of Frænkel-Mostowski models of first-order set theory (i.e., sets with atoms and permutations).

In this paper, we provide yet another formulation of the π-calculus with the aim of expressing generation of fresh names at every level (syntactic, semantic and implementative), and still keeping the metalogical overhead as low as possible. In fact, the calculus that we will present in Section 2, is just a conservative extension of the ordinary π-calculus with a new unary binding operator θ; for this reason, it is called the *$\pi\theta$-calculus*. This extension is suggested by the higher order presentations of the π-calculus as in [6, 10, 11, 14].

The new operator θ allows to explain "fresh" names as "locally θ-bound" names. A transition which needs a fresh name is rendered as a transition to a *θ-abstracted process*, i.e. where the fresh name is θ-bound. As we will see, many aspects of the treatment of the theory and metatheory of the π-calculus will benefit from this simple extension. Differently from the π-calculus, in the $\pi\theta$-calculus also actions are taken up-to α-conversion, thus yielding a *finitely branching* semantics w.r.t. fresh names. For example, while the π-process $(\nu y)\bar{x}y.P$ can evolve via $(\nu y)\bar{x}y.P \xrightarrow{\bar{x}(z)} P\{z/y\}$ for any fresh name z (and thus it is infinitely branching), the $\pi\theta$-process has just one move $(\nu y)\bar{x}y.P \xrightarrow{(\theta y)\bar{x}y} (\theta y)P$.

As we will see in Section 3, the encoding of both the syntax and the semantics of the $\pi\theta$-calculus in a logical framework based on Constructive Type Theory is direct and natural. In particular, all side conditions of the operational semantics are automatically dealt with by the metalanguage. With respect to previous encodings of π-calculus [6,10,11], we need just one reduction judgement (instead of two) and we do not need to introduce abstractions and concretions.

Bisimulation on the $\pi\theta$-calculus can be given without any constraint on bound names in labels. This allows for a direct re-use of the techniques in [1,2,19] for defining a coalgebraic semantics. In particular, the semantics of a process is finitely branching without being parametrized by the set of names of possible partners, as was the case in [10]. Moreover, our semantics is *finitary*, in the sense that any *finitary* process, i.e. with a bound degree of parallelism, gives rise to a finite set of descendant processes, up-to *vacuous* bound names (i.e. abstracted names which do not appear in the body of the process) and ordinary structural congruence.

In Section 4 we define the coalgebraic semantics within a category Alg^1_ω of *structured coalgebras*, following [16]. The algebra structure that we consider is induced by a countable family of unary operators $\{\rho_i\}_{i\in\omega}$. The fact that our coalgebraic semantics is in particular an algebra homomorphism allows us to derive interesting properties on *active names* of processes in the final model.

In order to get a truly finite representation for finitary processes, in Section 5 we introduce the notion of *θ-automaton*. This is the counterpart, in our second-order setting, of the notion of *History Dependent Automaton* of [16]. We associate to each $\pi\theta$-process a θ-automaton which is finite in case the original process is finitary. States of θ-automata are given by collapsing the *orbits* of processes under the action of vacuous θ-operators. We introduce a notion of bisimulation on the states of θ-automata. Bisimilarity between π-calculus processes can be (finitely) verified by checking the bisimilarity relation on the corresponding θ-automata.

Conclusions, related work, and directions for future work are in Section 6.

1 The π-calculus

In this section, we introduce briefly the π-calculus; see [15, 17] for more details. In particular, we introduce the syntax of the language, the *early* operational semantics, and the equivalence relation of *early* bisimilarity.

In the π-calculus there are only two primitive entities: *names* and *processes* (or *agents*). Let $\mathcal{N}$ be an infinite set of names, ranged over by x, y. The set of processes $\mathcal{P}$, ranged over by P, Q, are closed terms (w.r.t. process variables Z) defined by the abstract syntax:

$$P ::= 0 \mid \bar{x}y.P \mid x(y).P \mid \tau.P \mid (\nu x)P \mid Z \mid \mathrm{rec}\, Z.P \mid P_1|P_2 \mid [x = y]P$$

where the bound process variable Z must be guarded in $\mathrm{rec}\, Z.P$. The operators are listed in decreasing order of precedence. The *input prefix* operator $x(y)$ and the *restriction* operator (νy) bind the occurrences of y in $x(y).P$ and $(\nu y)P$ respectively. Thus, for each process P we can define the sets of its *free names* $fn(P)$, *bound names* $bn(P)$ and *names* $n(P) \triangleq fn(P) \cup bn(P)$. Processes are taken up-to α-equivalence, which is defined as expected. Capture-avoiding substitution of a single name y in place of x in P is denoted by $P\{y/x\}$.

We denote by $\mathcal{P}_X$, where X is a finite set of names, the subset of π-calculus processes whose free names are in X.

There is a plethora of slightly different labeled transition systems for the operational semantics of the π-calculus, see e.g. [15, 17, 21]. Here, we present the original one for early operational semantics [15]: the relation $\xrightarrow{\mu}$ is the smallest relation over processes satisfying the rules in Figure 1. (The right versions of rules PAR, COM and CLOSE have been omitted.)

The early operational semantics exploits four actions, defined by the syntax $(\mathcal{L} \ni)\ \mu ::= \tau \mid xy \mid \bar{x}y \mid \bar{x}(z)$. Their intuitive meaning is the following:

silent action: $P \xrightarrow{\tau} Q$ means that P can reduce itself to Q without interacting with other processes;

free output: $P \xrightarrow{\bar{x}y} Q$ means that P can reduce itself to Q emitting the name y on the channel x;

$$\frac{-}{x(z).P \xrightarrow{xy} P\{y/z\}} \quad \text{(IN)} \qquad \frac{-}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \quad \text{(OUT)}$$

$$\frac{-}{\tau.P \xrightarrow{\tau} P} \quad \text{(TAU)} \qquad \frac{P \xrightarrow{\overline{x}y} P'}{(\nu y)P \xrightarrow{\overline{x}(y)} P'}\, y \neq x \quad \text{(OPEN)}$$

$$\frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}(y)} Q'}{P|Q \xrightarrow{\tau} (\nu y)(P'|Q')}\, y \notin fn(P) \quad (\text{CLOSE}_l) \qquad \frac{P \xrightarrow{\mu} P'}{[x=x]P \xrightarrow{\mu} P'} \quad \text{(MATCH)}$$

$$\frac{P \xrightarrow{\mu} P'}{(\nu y)P \xrightarrow{\mu} (\nu y)P'}\, y \notin n(\mu) \quad \text{(RES)} \qquad \frac{P\{\text{rec}\, Z.P/Z\} \xrightarrow{\mu} P'}{\text{rec}\, Z.P \xrightarrow{\mu} P'} \quad \text{(UNFOLD)}$$

$$\frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q}\, bn(\mu) \cap fn(Q) = \emptyset \quad (\text{PAR}_l) \qquad \frac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\overline{x}y} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \quad (\text{COM}_l)$$

Fig. 1. Early Operational semantics of the π-calculus.

free input: $P \xrightarrow{xy} Q$ means that P can receive from the channel x the name y and then evolve into Q;

bound output: $P \xrightarrow{\overline{x}(z)} Q$ means that P can evolve into Q emitting on the channel x a name z, which is bound in P (but not in Q); only upon synchronization, z will be shared with the receiving agents and restricted again.

The functions $fn(\cdot)$ and $bn(\cdot)$ are extended to actions, by putting $fn(\bar{x}(z)) = \{x\}$, $fn(xy) = fn(\bar{x}y) = \{x, y\}$, $fn(\tau) = bn(\tau) = bn(xy) = bn(\bar{x}y) = \emptyset$, $bn(\bar{x}(z)) = \{z\}$. As usual, $n(\mu) \triangleq fn(\mu) \cup bn(\mu)$.

The τ and free input and free output actions are called *free*, the remaining ones are called *bound*. Note that actions are *not* taken up-to α-equivalence.

Definition 1 (Early Bisimilarity). *A symmetric relation $\mathcal{R}$ over π-calculus processes is an* early bisimulation *iff, for all processes P, Q, if $P\ \mathcal{R}\ Q$ then*

for each $P \xrightarrow{\mu} P'$ with $bn(\mu) \cap fn(P, Q) = \emptyset$ then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $P'\ \mathcal{R}\ Q'$.

The early bisimilarity $\sim$ *is the greatest early bisimulation.*

2 The $\pi\theta$-calculus

In this section, we introduce the *$\pi\theta$-calculus,* an extension of the π-calculus, where processes can be prefixed possibly by a finite sequence of the new binding operator θ. This new operator can be used to take care of the allocation of fresh names; essentially, it allows to model a *fresh* name using a *bound* name.

Syntax. The sets of $\pi\theta$-processes $\mathcal{P}^\theta$ and $\pi\theta$-actions $\mathcal{L}^\theta$ are defined as follows:

$$\mathcal{P}^\theta \triangleq \{(\theta x_1) \ldots (\theta x_n)P \mid P \in \mathcal{P},\ x_1, \ldots, x_n \in \mathcal{N},\ n \geq 0\}$$
$$\mathcal{L}^\theta \triangleq \{(\theta x_1) \ldots (\theta x_n)\mu \mid \mu \in \mathcal{L}_f \cup \mathcal{L}_b,\ x_1, \ldots, x_n \in \mathcal{N},\ n \geq 0\}$$

where $\mathcal{P}$ is the set of π-calculus processes, $\mathcal{L}_f$ is the set of *free actions*, i.e., τ, free input and free output, and $\mathcal{L}_b$ is the set of *bound actions*, i.e. bound

input, $x(y)$, and bound output, $\bar{x}(y)$, where $x(\ )$ and $\bar{x}(\ )$ bind y. By abuse of notation, P, Q and μ will range also over $\mathcal{P}^\theta$ and $\mathcal{L}^\theta$, respectively. We will use the abbreviations $(\theta\vec{x})P$ and $(\theta\vec{x})\mu$ for the process $(\theta x_1)\dots(\theta x_n)P$, and for the label $(\theta x_1)\dots(\theta x_n)\mu$, respectively, where P and μ are θ-free. The operator θ binds the occurrences of $x_1 \dots x_n$ in $(\theta\vec{x})P$ and in $(\theta\vec{x})\mu$. *Both* processes and labels are taken up-to α-equivalence; hence, without loss of generality, $x_1, \dots, x_n$ in $\vec{x}$ can always be assumed to be all distinct (e.g., $(\theta xx)P$ is the same as $(\theta xy)P\{y/x\}$).

The process $(\theta x)P$ can be viewed as the representation of a process abstraction obtained by instantiating P with a fresh name; the name which has to be fresh remains bound in P so that its freshness is guaranteed implicitly. In a sense, θ-abstractions resemble the λ-*abstractions* of the alternative presentations of the π-calculus in, e.g., [17, §5.5]. However, our aims are different; in fact, we do not have a notion of "application" (i.e., *concretion*).

For X a finite set of names, we denote by $\mathcal{P}^\theta_X$ and $\mathcal{L}^\theta_X$, the sets of $\pi\theta$-processes and $\pi\theta$-labels whose free names are in X, respectively. The set of *closed* $\pi\theta$-processes is $\mathcal{P}^\theta_\emptyset$. The occurrence (θx_i) in the process $(\theta x_1 \dots x_n)P$ is *vacuous* if $x_i \notin fn(P)$ or equivalently if the result of instantiating the occurrences of the name x_i in P with an arbitrary name produces always *the same* result.

Operational Semantics. The operational semantics of the $\pi\theta$-calculus is given by a family of relations. For X a finite set of names, the relation

$$\longrightarrow\!\!\!\!\!\rightarrow_X \subseteq \mathcal{P}^\theta_X \times \mathcal{L}^\theta_X \times \mathcal{P}^\theta_X$$

is defined as the smallest relation satisfying the rules in Figure 2. By definition, for any transition of the form $(\theta\vec{x})P \stackrel{(\theta\vec{y})\mu}{\longrightarrow\!\!\!\!\!\rightarrow_X} (\theta\vec{z})Q$, we have $fn((\theta\vec{y})\mu, (\theta\vec{z})Q) \subseteq fn((\theta\vec{x})P)$. Hence, if $(\theta\vec{x})P$ is closed, then X can be set to $\emptyset$.

Note that there are two input rules, IN and IN$^\theta$. In rule IN the bound name is instantiated with a "previously known" name z in X. Rule IN$^\theta$ takes care of the instantiation with a fresh name, by creating a new θ-bound name y. In this way, all π-calculus input transitions differing by the choice of the new name are collapsed (by α-rule) in a single transition, and the $\pi\theta$-system becomes finitely branching. As in rule IN$^\theta$, also in rule OPEN, the allocation of a fresh name is delegated to the constructor θ. The rules PAR and RES are duplicated, to take into account the case in which a θ-bound name appears in the target process.[3]

We remark that processes $(\nu x)P$ and $(\theta x)P$ behave differently, in general. Namely, rules RES and OPEN do not allow for output actions whose subject is exactly x, while the process $(\theta x)P$ could make an output transition under θ.

The following lemma clarifies the rôle of θ, and it is the counterpart of [15, Lemma 3] for the $\pi\theta$-calculus.

Lemma 1. *For all X finite, for all $(\theta\vec{x})P \in \mathcal{P}^\theta_X$:*

i) for all $(\theta\vec{x})Q, (\theta\vec{x})\mu$: $\quad (\theta\vec{x})P \stackrel{(\theta\vec{x})\mu}{\longrightarrow\!\!\!\!\!\rightarrow_X} (\theta\vec{x})Q$ *iff* $P \stackrel{\mu}{\longrightarrow\!\!\!\!\!\rightarrow_{X\cup\{\vec{x}\}}} Q$;

ii) for all $(\theta\vec{x}y)Q, (\theta\vec{x})\mu$: $\quad (\theta\vec{x})P \stackrel{(\theta\vec{x})\mu}{\longrightarrow\!\!\!\!\!\rightarrow_X} (\theta\vec{x}y)Q$ *iff* $P \stackrel{\mu}{\longrightarrow\!\!\!\!\!\rightarrow_{X\cup\{\vec{x}\}}} (\theta y)Q$.

[3] Strictly speaking, the side conditions "$z \in X$" of rule IN and "$y \notin fn(\mu)$" of rules RES and RES$^\theta$ are redundant, because they are always ensured by the type of $\longrightarrow\!\!\!\!\!\rightarrow_X$.

$$\frac{-}{\tau.P \xrightarrow{\tau}\!\!\!\!\rightarrow_X P} \quad \text{(TAU)} \qquad \frac{P \xrightarrow{\overline{x}y}\!\!\!\!\rightarrow_{X \uplus \{y\}} P'}{(\nu y)P \xrightarrow{\overline{x}(y)}\!\!\!\!\rightarrow_X (\theta y)P'} \quad \text{(OPEN)}$$

$$\frac{-}{x(y).P \xrightarrow{xz}\!\!\!\!\rightarrow_X P\{z/y\}}\, z \in X \quad \text{(IN)} \qquad \frac{P\{\mathrm{rec}\, Z.P/Z\} \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'}{\mathrm{rec}\, Z.P \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'} \quad \text{(UNFOLD)}$$

$$\frac{-}{x(y).P \xrightarrow{x(y)}\!\!\!\!\rightarrow_X (\theta y)P} \quad \text{(IN}^\theta\text{)} \qquad \frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'}{[x=x]P \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'} \quad \text{(MATCH)}$$

$$\frac{-}{\overline{x}y.P \xrightarrow{\overline{x}y}\!\!\!\!\rightarrow_X P} \quad \text{(OUT)} \qquad \frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_{X \uplus \{y\}} P'}{(\nu y)P \xrightarrow{\mu}\!\!\!\!\rightarrow_X (\nu y)P'}\, y \notin fn(\mu),\, \mu \in \mathcal{L}_f \quad \text{(RES)}$$

$$\frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'}{P|Q \xrightarrow{\mu}\!\!\!\!\rightarrow_X P'|Q}\, \mu \in \mathcal{L}_f \quad \text{(PAR}_l\text{)} \qquad \frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_{X \uplus \{y\}} (\theta z)P'}{(\nu y)P \xrightarrow{\mu}\!\!\!\!\rightarrow_X (\theta z)(\nu y)P'}\, y \notin fn(\mu) \quad \text{(RES}^\theta\text{)}$$

$$\frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_X (\theta x)P'}{P|Q \xrightarrow{\mu}\!\!\!\!\rightarrow_X (\theta x)P'|Q} \quad \text{(PAR}_l^\theta\text{)} \qquad \frac{P \xrightarrow{x(y)}\!\!\!\!\rightarrow_X (\theta y)P' \quad Q \xrightarrow{\overline{x}(y)}\!\!\!\!\rightarrow_X (\theta y)Q'}{P|Q \xrightarrow{\tau}\!\!\!\!\rightarrow_X (\nu y)P'|Q'} \quad \text{(CLOSE}_l\text{)}$$

$$\frac{P \xrightarrow{xy}\!\!\!\!\rightarrow_X P' \quad Q \xrightarrow{\overline{x}y}\!\!\!\!\rightarrow_X Q'}{P|Q \xrightarrow{\tau}\!\!\!\!\rightarrow_X P'|Q'} \quad \text{(COM}_l\text{)} \qquad \frac{P \xrightarrow{\mu}\!\!\!\!\rightarrow_{X \uplus \{x\}} Q}{(\theta x)P \xrightarrow{(\theta x)\mu}\!\!\!\!\rightarrow_X (\theta x)Q} \quad \text{(THETA)}$$

Fig. 2. Early Operational semantics of the $\pi\theta$-calculus.

We can draw a precise correspondence between π- and $\pi\theta$-derivations:

Proposition 1. *For all X finite, for all $\vec{z}$ not in X, and $P \in \mathcal{P}_{X \cup \{\vec{z}\}}$, $x, y \in \mathcal{N}$:*

- $P \xrightarrow{\tau} Q$ *iff* $(\theta\vec{z})P \xrightarrow{(\theta\vec{z})\tau}\!\!\!\!\rightarrow_X (\theta\vec{z})Q$;
- $P \xrightarrow{\bar{x}y} Q$ *iff* $(\theta\vec{z})P \xrightarrow{(\theta\vec{z})\bar{x}y}\!\!\!\!\rightarrow_X (\theta\vec{z})Q$;
- $P \xrightarrow{xy} Q$ *iff* $\Big($*if* $y \in X \cup \{\vec{z}\}$ *then* $(\theta\vec{z})P \xrightarrow{(\theta\vec{z})xy}\!\!\!\!\rightarrow_X (\theta\vec{z})Q$ *else* $(\theta\vec{z})P \xrightarrow{(\theta\vec{z})x(y)}\!\!\!\!\rightarrow_X (\theta\vec{z}y)Q\Big)$;
- $P \xrightarrow{\bar{x}(y)} Q$ *iff* $(\theta\vec{z})P \xrightarrow{(\theta\vec{z})\bar{x}(y)}\!\!\!\!\rightarrow_X (\theta\vec{z}y)Q$.

The proof of Proposition 1 is straightforward by mutual induction on the structure of derivations. In particular, when $\vec{z}$ is empty:

Corollary 1. *For all X finite, for all $P \in \mathcal{P}_X$, for all $x, y \in \mathcal{N}$:*

- $P \xrightarrow{\tau} Q$ *iff* $P \xrightarrow{\tau}\!\!\!\!\rightarrow_X Q$;
- $P \xrightarrow{\bar{x}y} Q$ *iff* $P \xrightarrow{\bar{x}y}\!\!\!\!\rightarrow_X Q$;
- $P \xrightarrow{xy} Q$ *iff* $\Big($*if* $y \in X$ *then* $P \xrightarrow{xy}\!\!\!\!\rightarrow_X Q$ *else* $P \xrightarrow{x(y)}\!\!\!\!\rightarrow_X (\theta y)Q\Big)$;
- $P \xrightarrow{\bar{x}(y)} Q$ *iff* $P \xrightarrow{\bar{x}(y)}\!\!\!\!\rightarrow_X (\theta y)Q$.

The relations $\longrightarrow\!\!\!\!\rightarrow_X$ can be seen as a family of coherent "approximations" of the usual early operational semantics. We can recover this semantics by taking the union of all approximations: for μ action of the $\pi\theta$-calculus, we define $\xrightarrow{\mu}\!\!\!\!\rightarrow \triangleq \bigcup_X \xrightarrow{\mu}\!\!\!\!\rightarrow_X$. Of course, we could drop safely the X parameter and consider all the transition systems simultaneously, without any other consequence but that the operational semantics would then be finitely branching *only* w.r.t. fresh

names. At the moment, each $\longrightarrow\!\!\!\!\!\rightarrow_X$ is *truly* finitely branching, because the names which can be chosen in the rule IN must belong to X, which is finite. We prefer to keep the X parameter, in order to provide a sharper analysis of the system.

Bisimulation. We can now introduce a notion of bisimulation on $\pi\theta$-processes, which provides an alternative characterization of early bisimilar processes.

Definition 2 (Early θ-bisimilarity). *Let X be a finite set of names. A symmetric relation $\mathcal{R}_X \subseteq \mathcal{P}^\theta_X \times \mathcal{P}^\theta_X$ is an* early θ-bisimulation at stage X *iff, for all $P, Q \in \mathcal{P}^\theta_X$ processes, $P\ \mathcal{R}_X\ Q$ implies:*

– *if $P \stackrel{\mu}{\longrightarrow\!\!\!\!\!\rightarrow}_X P'$, then there exists Q' such that $Q \stackrel{\mu}{\longrightarrow\!\!\!\!\!\rightarrow}_X Q'$ and $P'\ \mathcal{R}_X\ Q'$.*

The early θ-bisimilarity at stage X, $\approx_X$, *is the greatest early θ-bisimulation at stage X. The* early θ-bisimilarity $\approx$ *is defined as $\approx \triangleq \bigcup_X \approx_X$.*

Notice that the notion of early θ-bisimilarity depends, for generic processes, on their free names. Again we could disregard X completely. Anyway, for θ-closed processes, any reference to names disappears altogether. What is more significant, however, is that for all (possibly open) processes, the side condition on the freshness of names necessary for bound output in Definition 1 disappears, being implicit in the fact that the new name is bound by θ in P', Q'. The price to pay is that each time a fresh name is needed, an extra (possibly vacuous) θ is generated, and the set of processes reached during the evolution of a finitary process is finite, only *up-to vacuous θ's*.

Example 1. Let us consider the recursive process $P = \mathrm{rec}\, Z.(\nu y).\bar{x}y.Z \in \mathcal{P}_X$, where $X = \{x\}$. In the π-calculus, the process P can evolve into itself, i.e. $P \stackrel{\bar{x}(y)}{\longrightarrow} P \stackrel{\bar{x}(y)}{\longrightarrow} \ldots$ while, in the $\pi\theta$-calculus, P can evolve as follows:

$$P \stackrel{\bar{x}(y_0)}{\longrightarrow\!\!\!\!\!\rightarrow}_X (\theta y_0)P \stackrel{(\theta y_0)\bar{x}(y_1)}{\longrightarrow\!\!\!\!\!\rightarrow}_X (\theta y_0 y_1)P \stackrel{(\theta y_0 y_1)\bar{x}(y_2)}{\longrightarrow\!\!\!\!\!\rightarrow}_X (\theta y_0 y_1 y_2)P \longrightarrow\!\!\!\!\!\rightarrow_X \ldots$$

Notice that the states reached by P after a finite number of transition steps differ by a finite number of vacuous θ's.

The following lemma can be viewed as the "higher order" version of Lemma 6 of [15], and it is instrumental to prove Theorem 1 below.

Lemma 2. *Let $(\theta\vec{x})P, (\theta\vec{x})Q \in \mathcal{P}^\theta_X$. Then $(\theta\vec{x})P \approx_X (\theta\vec{x})Q$ iff $P \approx_{X\cup\{\vec{x}\}} Q$.*

As a main correspondence result, θ-bisimilarity is a conservative extension of usual bisimilarity:

Theorem 1. *Let $P, Q \in \mathcal{P}$. Then $P \sim Q$ iff $P \approx Q$.*

Proof. Both directions are proved by coinduction, using Proposition 1.
($\Rightarrow$) We prove that the relation
$\mathcal{R} = \{((\theta x_1 \ldots x_n)P, (\theta x_1 \ldots x_n)Q \mid n \geq 0, x_1, \ldots, x_n \in \mathcal{N}\ \wedge\ P \sim Q\}$
is an early θ-bisimulation at stage X, for $X \supseteq fn(P, Q)$.
($\Leftarrow$) Using Lemma 2, we prove that the relation $\mathcal{R} = \{(P, Q) \mid P \approx Q\}$ is an early bisimulation. □

Remark 1. In the light of Lemma 2, one could wonder whether it is possible to simplify our notion of early θ-bisimulation, by getting rid of redundant vacuous θ's. This would allows us to overcome the problem highlighted by Example 1. But even if we restrict ourselves to processes whose occurrences of names are all active[4], independent elimination of vacuous θ's is not safe, as we can see from the counterexample below.

Example 2. Let $P = (\nu x)\bar{w}x.(\nu y)\bar{x}y.\bar{u}x.0$, $Q = (\nu x)\bar{w}x.(\nu y)\bar{x}y.\bar{u}y.0$. Then P and Q are *not* early bisimilar, because the last action of P consists in communicating the first extruded name, while Q communicates the second extruded name. But P and Q turn out to be erroneously equated if we eliminate vacuous θ's. Namely, after two transition steps P reduces to $(\theta xy)\bar{u}x.0$, and Q reduces to $(\theta xy)\bar{u}y.0$, but, since θx is vacuous in the first process, while θy is vacuous in the latter, we reduce ourselves to considering the pair of processes $(\theta x)\bar{u}x$ and $(\theta y)\bar{u}y$, which turn out to be α-equivalent and thus bisimilar.

3 Implementation of the $\pi\theta$-calculus in Coq

In this section we sketch an implementation of the $\pi\theta$-calculus in Logical Framework based on a constructive Type Theory, such as the Edinburgh LF or Coq [8,12]. This is a simplification of previous formalizations of the π-calculus [10,11], because the encoding of operational semantics will not need an auxiliary transition system for bound actions, i.e., actions leading to processes with fresh names.

Following the HOAS methodology, we can delegate to the metalogic all the bookkeeping aspects related to name generation and freshness. At the syntactic level, α-conversion is inherited; at the semantic level, side conditions disappear, and even the indexing of the transition relation is dealt with by the very shape of the typing judgement.

Syntax. Names and variables will be represented by LF variables of specific types `Name` and `Var`, without constructors. θ-free processes $\mathcal{P}$ are represented by terms of type `Proc`, which is a subtype of type `TProc` representing the whole $\mathcal{P}^\theta$. Constructors of these two types are as usual; in particular, all binding constructors (input, restriction, recursion and θ) are represented by second-order term constructors; hence α-conversion and capture avoiding substitution of names and variables are inherited from the metalanguage.

```
Variable Name,Var : Set.
Inductive Proc: Set :=  nil     : Proc
  | output  : Name -> Name -> Proc -> Proc
  | input   : Name -> (Name -> Proc) -> Proc
  | tau     : Proc -> Proc
  | nu      : (Name -> Proc) -> Proc
```

[4] An occurence of a name is *active* in P if it appears explicitly in an action in the evolution of P.

```
  | inVar   : Var -> Proc
  | rec     : (Var -> Proc) -> Proc
  | par     : Proc -> Proc -> Proc
  | match   : Name -> Name -> Proc -> Proc.
Coercion inVar : Var >-> Proc.
Inductive TProc : Set :=
   inProc : Proc -> TProc   |   theta : (Name -> TProc) -> TProc.
Coercion inProc : Proc >-> TProc.
```

For $X = \{x_1, \ldots, x_n\}$ a finite set of names, we will denote by Γ_X the typing environment `x1:Name,...,xn:Name`, and by $\texttt{Proc}_X$ the canonical forms `P` of type `Proc` such that $\Gamma_X \vdash \texttt{P} : \texttt{Proc}$; similarly for `TProc`. Then:

Proposition 2. *For each $X \subset \mathcal{N}$ finite, there are two compositional bijections $\epsilon_X : \mathcal{P}_X \to \texttt{Proc}_X$, and $\epsilon^\theta_X : \mathcal{P}^\theta_X \to \texttt{TProc}_X$.*

In the following, both encoding functions will be denoted by ϵ_X.

Semantics. In order to implement the early operational semantics in Figure 2, we need to introduce the syntactic sorts of free and bound actions `FAct`, `BAct`, and θ-actions `TAct`.

```
Inductive FAct : Set := ftau : FAct
  | fout : Name -> Name -> FAct | fin : Name -> Name -> FAct.
Inductive BAct : Set :=  bout : Name -> BAct  | bin : Name -> BAct.
Inductive TAct : Set :=  inFAct : FAct -> TAct
  |  inBAct : BAct -> TAct | theta_a : (Name -> TAct) -> TAct.
Coercion inFAct : FAct >-> TAct.
Coercion inBAct : BAct >-> TAct.
```

Let us denote again by ϵ_X the obvious encoding function for actions.

The transition relation $P \xrightarrow{\mu}_X Q$ is represented by a predicate `trans : TProc -> TAct -> TProc -> Prop`. All rules are encoded straightforwardly; see Appendix A for the complete code. In particular, side conditions disappear, because they are automatically dealt with by the metalanguage features. For instance, rules (RES) and (RES$^\theta$) are encoded respectively as follows:

```
fRES  : (P,Q:Name->Proc)(a:FAct)
        ((y:Name)(trans (P y) a (Q y))) -> (trans (nu P) a (nu Q))
bRES  : (P:Name->Proc)(P':Name->Name->Proc)(a:BAct)
        ((y:Name)(trans (P y) a (theta [z:Name](P' z y)))) ->
        (trans (nu P) a (theta [z:Name](nu (P' z))))
```

The local name `y` is automatically fresh, different from any other existing name, and in particular different from any name occurring in `a`. Notice that we do not need to parametrize `trans` by the set of names, because this is automatically represented by the variables of type `name` declared in the proof context:

Proposition 3. *Let $X \subset \mathcal{N}$ be a finite set of names, and $P, Q \in \mathcal{P}^\theta_X$. Then:*

$$P \xrightarrow{\mu}_X Q \iff \exists d. \quad \Gamma_X \vdash d : (\texttt{trans}\ \epsilon_X(P)\ \epsilon_X(\mu)\ \epsilon_X(Q))$$

4 Finitary Coalgebraic Semantics

In this section, we capitalize on the results of the previous sections, in order to give a coalgebraic description of early bisimilarity which is both *not* parametrized on sets of names as well as *finitary* (up-to structural congruence and vacuous bound names) for finitary processes. We focus on *closed* $\pi\theta$-processes, since, by Lemma 2, bisimilarity of (possibly open) $\pi\theta$-processes can always be reduced to bisimilarity of θ-closed processes.

Recall that a T-coalgebra on a category $\mathbf{C}$ (e.g. $\mathcal{S}et$), where T is an endofunctor, is a pair (A, α), where A is an object of $\mathbf{C}$ and $\alpha : A \to T(A)$ is an arrow of $\mathbf{C}$. A T-coalgebra morphism $h : (A, \alpha) \to (B, \beta)$ is an arrow $h : A \to B$ of $\mathbf{C}$ (e.g., a function when $\mathbf{C} = \mathcal{S}et$), such that $\beta \circ h = T(h) \circ \alpha$.

According to the final semantics approach [1,2,19], the operational semantics of a language is represented as a *T-coalgebra* for a suitable endofunctor T. If the functor is "well-behaved," then there exists a *final T-coalgebra*, say $\Omega = (\Omega, \alpha_\Omega)$. Moreover, for any T-coalgebra (A, α), the unique arrow $\mathcal{M} : (A, \alpha) \to (\Omega, \alpha_\Omega)$ induces an equivalence on A which can be characterized as the union of all *T-bisimulations* on (A, α)–a T-bisimulation on (A, α) is the categorical counterpart of the ordinary notion of bisimulation: it is a relation $\mathcal{R} \subseteq A \times A$ for which there exists an arrow $\gamma : \mathcal{R} \to T(\mathcal{R})$, such that the projections $\pi_1,\ \pi_2$ can be lifted to T-coalgebra morphisms from $(\mathcal{R}, \gamma)$ to (A, α).

In order to take advantage of the algebraic structure of θ-operators and be able to capture the number of active names in processes, we work in a category of *structured* coalgebras. We consider coalgebras over the category Alg^1_ω of *1-ary ω-algebras*, which are algebras with an infinite family of unary operators.

Definition 3 ($^1_\omega$-algebra). *A* 1-ary ω-algebra ($^1_\omega$-algebra) *is a set A, the* carrier*, with an infinite family $\rho^A = \{\rho^A_i\}_{i\in\omega}$ of unary operators $\rho^A_i : A \to A$, closed under composition. The category of $^1_\omega$-algebras Alg^1_ω is defined as follows:*

- *objects are $^1_\omega$-algebras*
- *morphisms $f : (A, \rho^A) \longrightarrow (B, \rho^B)$ are all functions $f : A \to B$ in $\mathcal{S}et$, which preserve the algebraic structure, i.e. $f \circ \rho^A_i = \rho^B_i \circ f$ for all $i \in \omega$.*

The set $\mathcal{P}^\theta_\emptyset$ of closed $\pi\theta$-calculus processes can be endowed with a structure of a 1-ary ω-algebra:

Proposition 4. *For $n \in \omega$, let us denote by $\pi_n : n \to n$ a permutation of $\{1, \ldots, n\}$. For $m \geq n$, let $\pi_{n_{|m}}$ be the extension of π_n to a permutation on m with the identity on $\{n+1, \ldots, m\}$. For $|\vec{x}| = n$, we let $\pi_n[\vec{x}] \triangleq x_{\pi_n 1} \ldots x_{\pi_n n}$. The* prefix permutation *operator $\rho_{\pi_n} : \mathcal{P}^\theta_\emptyset \to \mathcal{P}^\theta_\emptyset$ is defined as follows:*

$$\rho_{\pi_n}((\theta\vec{x})P) \triangleq \begin{cases} (\theta\pi_{n_{|\vec{x}|}}[\vec{x}])P & \text{if } |\vec{x}| \geq n \\ (\theta\vec{x})P & \text{otherwise .} \end{cases}$$

Composition $\rho_{\pi_n} \circ \rho_{\pi_m}$ of prefix permutations is defined as $\rho_{\pi_{n_{|m}}} \circ \rho_{\pi_m}$, when $m = \max(m, n)$, and as $\rho_{\pi_n} \circ \rho_{\pi_{m_{|n}}}$ otherwise. Let $\{\rho_i\}_{i\in\omega}$ be an enumeration of the set $\rho = \{\rho_{\pi_n} \mid \pi_n : n \to n,\ n \in \omega\}$, then $(\mathcal{P}^\theta_\emptyset, \{\rho_i\}_{i\in\omega})$ is a $^1_\omega$-algebra.

The operational semantics induces a structure of coalgebra on the ω-algebra of closed $\pi\theta$-processes for a functor similar to the one used for CCS:

Definition 4. *Let $T : Alg^1_\omega \to Alg^1_\omega$ be the functor defined by the canonical extension to arrows of the function:*

$$T(A, \rho^A) \triangleq (\wp_{\mathrm{f}}(\mathcal{L}^\theta \times A), \rho^{\wp_{\mathrm{f}}(\mathcal{L}^\theta \times A)}) \ ,$$

where, for any $u \in \wp_{\mathrm{f}}(\mathcal{L}^\theta \times A)$, $\pi_n : n \to n$,

$$\rho^{\wp_{\mathrm{f}}(\mathcal{L}^\theta \times A)}_{\pi_n}(u) \ = \ \{((\theta \pi_{n|_{|\vec{x}|}}[\vec{x}])\mu, \rho_{\pi_n} a) \mid ((\theta\vec{x})\mu, a) \in u\} \ .$$

Proposition 5. *Let $\alpha : (\mathcal{P}^\theta_\emptyset, \rho) \to T(\mathcal{P}^\theta_\emptyset, \rho)$ be defined as follows:*

$$\alpha(P) \triangleq \{(\mu, Q) \mid P \xrightarrow{\mu}\!\!\!\!\!\rightarrow_\emptyset Q\}$$

Then, $\mathcal{C}_{\pi\theta} \triangleq (\mathcal{P}^\theta_\emptyset, \rho, \alpha)$ is a T-coalgebra.

Since T is a lifting of the corresponding polynomial functor, by Theorem 7 of Appendix B, we have:

Proposition 6. *The functor T has a final coalgebra $\Omega = (\Omega, \rho^\Omega, \alpha_\Omega)$.*

Remark 2. One may wonder whether Proposition 5 still holds if we consider other constructors in the algebraic structure beside ρ's. The point is that α has to be a morphism between $^1_\omega$-algebras, i.e. it has to respect the algebraic structures. This holds for $\nu, |$, thus providing an alternative proof of the fact that bisimilarity is a congruence w.r.t. $\nu, |$. But it does not hold in the case of input prefix, of course. For example, let us consider the operator $\iota_{zx}(\cdot) = z(x).\cdot$ acting on processes as follows: $\iota_{zx}((\theta\vec{y})P) \triangleq (\theta\vec{y}z)z(x).P$. If $P = (\theta yw)\bar{y}w$, then $\alpha(\iota_{zx}(P)) \neq T(\iota_{zx})(\alpha(P))$, whatever is the action of ι_{zx} on labels.

Using Lemma 2 and Theorem 1, we can easily prove that:

Theorem 2. *Let $P, Q \in \mathcal{P}$ be such that $(\theta\vec{x})P, (\theta\vec{x})Q \in \mathcal{P}^\theta_\emptyset$. Then $P \sim Q$ iff there exists a T-bisimulation, $\mathcal{R}$, on the coalgebra $\mathcal{C}_{\pi\theta}$ such that $(\theta\vec{x})P \ \mathcal{R} \ (\theta\vec{x})Q$.*

The following proposition characterizes T-bisimilarity by finality, and hence, by Theorem 2, also bisimilarity on π-calculus processes.

Proposition 7. *The equivalence induced by the unique morphism $\mathcal{M} : \mathcal{C}_{\pi\theta} \longrightarrow \Omega$ coincides with the union of all T-bisimulations on the T-coalgebra $\mathcal{C}_{\pi\theta}$.*

Finally, we can put to use the $^1_\omega$-algebraic structure of coalgebras. In [16], the richer structure of coalgebras given by permutation algebras was used to show that the support of the interpretation of a π-calculus process in the final model amounts exactly to the active names of the process. In our setting we can obtain a similar result:

Proposition 8. *For $(\theta\vec{x})P \in \mathcal{P}^\theta_\emptyset$, the family $\{\rho^\Omega_i(\mathcal{M}(P))\}_i$ has at most $n!/k!$ distinct elements, where $n = |\vec{x}|$ and k is the number of active names of P.*

Proof. (Outline) The action of ρ-operators commutes with the final semantics. By definition of active names, swapping non-active names in the θ-prefix does not change the bisimilarity class of the process. Therefore, the number of distinct elements in the family $\{\rho^\Omega_i(\mathcal{M}(P))\}_i$ is bounded by the number of different permutations of n objects, k of which are equal. □

5 θ-automata

In this section, drawing inspiration from [16], we introduce a notion of automaton, called *θ-automaton*, for representing in a *finite* way the evolution of *finitary* $\pi\theta$-processes. These are processes whose descendants have a bounded number of possible parallel actions (*degree of parallelism*):

Definition 5 (Finitary Process). *The* degree of parallelism $\deg(P)$ *of a $\pi\theta$-process P is defined as follows (for π a generic action prefix):*

$$\deg(0) = \deg(Z) \triangleq 0 \qquad \deg(\pi.P) \triangleq 1 \qquad \deg(P \mid Q) \triangleq \deg(P) + \deg(Q)$$
$$\deg((\nu x)P) = \deg((\theta x)P) = \deg([x = y]P) = \deg(\mathrm{rec}\, Z.P) \triangleq \deg(P)$$

A process $P \in \mathcal{P}^\theta_X$ is finitary *if* $\max\{\deg(P') \mid P \xrightarrow{\mu_1}\!\!\!\!\rightarrow_X \ldots \xrightarrow{\mu_i}\!\!\!\!\rightarrow_X P'\} < \infty$. *For P finitary, let us denote the maximum degree of any descendant*[5] *of P by* $\overline{\deg}(P)$.

Structurally congruent $\pi\theta$-processes will be represented by the same state:

Definition 6 (Structural Congruence). *The* structural congruence $\equiv$ *on π-calculus processes is the smallest congruence that satisfies the following:*

(par) $P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
(res) $(\nu x)0 \equiv 0 \qquad (\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$, *if* $x \notin fn(P)$
$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \qquad (\nu x)\pi.P \equiv \pi.(\nu x)P$, *if* $x \notin fn(\pi)$
(match) $[x = x]P \equiv P \qquad [x = y]0 \equiv 0$
(unfold) $\mathrm{rec}\, Z.P \equiv P\{\mathrm{rec}\, Z.P/Z\}$

where π stands for a generic action/matching prefix.

The $\pi\theta$-processes $(\theta\vec{x})P$, $(\theta\vec{x})Q$ are structurally congruent *(also denoted by $\equiv$) if and only if $P \equiv Q$.*

For each class $\mathcal{S} \subseteq \mathcal{P}^\theta$ of congruent processes, let us fix a representative process P such that $|fn(P)| = min\{|fn(Q)| \mid Q \in \mathcal{S}\}$.

Moreover, since processes differing by vacuous θ's will be collapsed in the same state of the θ-automaton, we need to introduce a *canonical* representative for classes of congruent processes together with all processes differing by vacuous θ's. For the sake of simplicity, but without loss of generality by Lemma 2, we introduce θ-automata only for *closed* $\pi\theta$-processes.

Definition 7. *A $\pi\theta$-process $(\theta x_1 \ldots x_n)P \in \mathcal{P}^\theta_\emptyset$ is* canonical *if it is the representative of a $\equiv$-class and $x_i \in fn(P)$, for all $1 \leq i \leq n$. Let* $\mathrm{can}(\mathcal{P}^\theta_\emptyset)$ *denote the set of canonical $\pi\theta$-processes.*

Let $\|\cdot\| : \mathcal{P}^\theta_\emptyset \to \mathrm{can}(\mathcal{P}^\theta_\emptyset)$ be defined by $\|(\theta x_1 \ldots x_n)P\| \triangleq (\theta y_1 \ldots y_m)P'$, where P' is the representative of the equivalence class of P, $x_1 \ldots x_n = \vec{z}_0 y_1 \vec{z}_1 \ldots y_m \vec{z}_m$, $|\vec{z}_i| \geq 0$, and $\{y_1, \ldots, y_m\} = fn(P')$. We define the reindexing[6] *$\xi((\theta x_1 \ldots x_n)P) \in \mathbb{M}(n, m)$ as follows: $\xi((\theta x_1 \ldots x_n)P)(i) = j \iff x_i = y_j$.*

[5] A process Q is a *descendant* of P if $P \longrightarrow\!\!\!\!\rightarrow^*_X Q$, where $\longrightarrow\!\!\!\!\rightarrow^*_X$ is the reflexive and transitive closure of $\longrightarrow\!\!\!\!\rightarrow_X$.

[6] Where $\mathbb{M}(n, m)$ denotes the set of partial strict monotone functions from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$.

For a canonical process $(\theta\vec{x})P$*, we define its* orbit *as the set* $\mathrm{orbit}((\theta\vec{x})P) \triangleq \{(\theta\vec{z}_0 x_1 \ldots \vec{z}_{n-1} x_n \vec{z}_n)P' \in \mathcal{P}^\theta_\emptyset \mid P' \equiv P \ \wedge \ |\vec{z}_0|, \ldots, |\vec{z}_n| \geq 0 \ \wedge \ \forall i, j. x_j \notin \vec{z}_i\}$.

Definition 8 (θ-automaton). *Let* $P \in \mathcal{P}^\theta_\emptyset$*. The* θ-automaton $\mathcal{A}_P$ *induced by* P *is the triple* $(\mathcal{S}, \|P\|, \mapsto)$*, where:*

- $\mathcal{S}$ *is the set of* states*. Each state is the orbit of the canonical process corresponding to a descendant of* P*, and it is denoted by the canonical representative itself.*
- $\|P\|$ *is the* initial state*.*
- $\mapsto \subseteq \mathcal{S} \times \mathcal{L}^\theta \times \mathcal{S}$ *is the* transition relation *defined by:*

$$P_1 \stackrel{\mu}{\mapsto} P_2 \text{ iff there exists } P_2' \text{ such that } P_1 \stackrel{\mu}{\twoheadrightarrow}_\emptyset P_2' \text{ and } \|P_2'\| = P_2 \ .$$

In order to prove the fundamental Theorem 3 below, which motivates the notion of θ-automaton, we first need the following technical definition.

Definition 9. *Let* P *be a* $\pi\theta$*-process (possibly with free process variables). The* set of subprocesses *of* P *is defined as* $sub(P) \triangleq \{P\} \cup sub'(P)$*, where:*

$$sub'(0) = sub'(Z) \triangleq \emptyset \qquad sub'(P|Q) \triangleq sub(P) \cup sub(Q)$$
$$sub'(\pi.P) = sub'([x = y]P) = sub'((\nu x)P) = sub'((\theta x)P) \triangleq sub(P)$$
$$sub'(\mathrm{rec}\, Z.P) \triangleq \{Q\{\mathrm{rec}\, Z.P/Z\} \mid Q \in sub(P)\}$$

Lemma 3. *If* $P \in \mathcal{P}^\theta$*, then the set* $sub(P)$ *is finite, and for all* $Q \in sub(P)$*:* $sub(Q) \subseteq sub(P)$*.*

Theorem 3. *Let* $(\theta\vec{x})P \in \mathcal{P}^\theta_\emptyset$ *be a finitary process. Then* $\mathcal{A}_{(\theta\vec{x})P}$ *is finite.*

Proof. Let $n_0 = \overline{\deg}(P)$, and Q be any descendant of $(\theta\vec{x})P$. Then, by definition of $\equiv$, either Q admits a canonical form $Q \equiv (\theta\vec{z})(\nu\vec{y})(Q_1| \ldots |Q_n)$, where $n \leq n_0$, and Q_i are *sequential* processes (i.e., non-null processes whose top operator is either an action prefix or a non-trivial matching) or $Q \equiv (\theta\vec{z})0$. Moreover, by the definition of the transition system, each component Q_i is a subprocess of P, up-to-$\equiv$ and (possibly non injective) name substitution, i.e. $Q_i \equiv P_i\sigma$, for a name substitution σ, and P_i is a subprocess of P. Since the subprocesses of P are finite, then there are only finitely many possible Q_i, up-to bijective name substitutions. Moreover, the number of free names in Q_i is bounded by the number of (either free or bound) names in P, $n(P)$, and hence the number of non vacuous ν's and θ's in $(\theta\vec{z})(\nu\vec{y})(Q_1 \mid \ldots \mid Q_n)$ is bounded by $n_0 \times |n(P)|$. Hence, the number of descendants Q of a finitary process is finite, up-to-$\equiv$ and vacuous θ's. □

One can easily check that the θ-automaton corresponding to (the θ-closure of) the recursive process of Example 1 is finite (actually it consists of exactly one state and one transition edge).

One could develop a complete theory of θ-automata, and recover classical results, such as minimalization. But here we shall only investigate how to use θ-automata in order to devise an effective procedure for establishing bisimilarity

of $\pi\theta$-processes. Given two processes, in order to use the induced θ-automata to check bisimilarity, we need to keep track, at each step, of the correspondence between θ-bound names in the canonical processes (Example 2 shows that elimination of vacuous θ's could otherwise compromise bisimilarity). To this aim, we use a finitary reindexing function, mapping the positions of variables which have to be identified in the two processes. Let us denote by $\mathbb{M}$ the set $\bigcup_{n,m\in\omega}\mathbb{M}(n,m)$:

Definition 10 (Indexed Bisimilarity). *Let $\mathcal{A}=(\mathcal{S},P_0,\mapsto)$, $\mathcal{A}'=(\mathcal{S}',Q_0,\mapsto')$ be θ-automata. An* indexed bisimulation *$\mathcal{R}\subseteq\mathcal{S}\times\mathbb{M}\times\mathcal{S}'$ is a relation such that, for $(\theta\vec{x})P\in\mathcal{S}$, $(\theta\vec{y})Q\in\mathcal{S}'$, for $f\in\mathbb{M}(|\vec{x}|,|\vec{y}|)$, if $((\theta\vec{x})P,f,(\theta\vec{y})Q)\in\mathcal{R}$ then: let $dom(f)=\{i_1,\ldots,i_k\}$, $\vec{x}=\vec{u}_0x_{i_1}\vec{u}_1\ldots x_{i_k}\vec{u}_k$, $\vec{y}=\vec{v}_0y_{f(i_1)}\vec{v}_1\ldots y_{f(i_k)}\vec{v}_k$, $\vec{x}\cap\vec{y}=\emptyset$, $\vec{z}=\vec{u}_0\vec{v}_0x_{i_1}\vec{u}_1\vec{v}_1\ldots x_{i_k}\vec{u}_k\vec{v}_k$,*

- *if $(\theta\vec{z})P\stackrel{(\theta\vec{z})\mu}{\longrightarrow\!\!\!\!\!\!\rightarrow}_{\emptyset}(\theta\vec{z})P'$, then there exists $(\theta\vec{z})Q'$ such that*
  - $(\theta\vec{z})Q\{x_{i_1}/y_{f(i_1)},\ldots,x_{i_k}/y_{f(i_k)}\}\stackrel{(\theta\vec{z})\mu}{\longrightarrow\!\!\!\!\!\!\rightarrow}_{\emptyset}(\theta\vec{z})Q'$
  - $(\|(\theta\vec{z})P'\|,f',\|(\theta\vec{z})Q')\|\in\mathcal{R}$, *where* $f'=\xi((\theta\vec{z})Q')\circ(\xi((\theta\vec{z})P'))^{-1}$.
- *if $(\theta\vec{z})P\stackrel{(\theta\vec{z})\mu}{\longrightarrow\!\!\!\!\!\!\rightarrow}_{\emptyset}(\theta\vec{z}\vec{z}')P'$, then there exists $(\theta\vec{z}\vec{z}')Q'$ such that*
  - $(\theta\vec{z})Q\{x_{i_1}/y_{f(i_1)},\ldots,x_{i_k}/y_{f(i_k)}\}\stackrel{(\theta\vec{z})\mu}{\longrightarrow\!\!\!\!\!\!\rightarrow}_{\emptyset}(\theta\vec{z}\vec{z}')Q'$
  - $(\|(\theta\vec{z}\vec{z}')P'\|,f',\|(\theta\vec{z}\vec{z}')Q')\|\in\mathcal{R})$, *where* $f'=f_1\cup f_2$, $f_1=\xi((\theta\vec{z}\vec{z}')Q')\circ(\xi((\theta\vec{z}\vec{z}')P'))^{-1}$ *and*

$$f_2=\begin{cases}\{(\max(\mathrm{dom}(f_1))+1,\max(\mathrm{cod}(f_1))+1)\} & \text{if } z'\in fn(P')\cap fn(Q')\\ \emptyset & \text{otherwise}\end{cases}$$

- $((\theta\vec{y})Q,f^{-1},(\theta\vec{x})P)\in\mathcal{R}$.

The indexed bisimilarity, *$\simeq$, is the greatest indexed bisimulation. We say that the automata $\mathcal{A}$ and $\mathcal{A}'$ are f-*bisimilar *if $(P_0,f,Q_0)\in\simeq$, for some $f\in\mathbb{M}(|\vec{x}|,|\vec{y}|)$.*

Using Lemma 2, one can prove that:

Theorem 4. *Let $P,Q\in\mathcal{P}_{\{x_1,\ldots,x_n\}}$. Then, $P\sim Q$ iff $\mathcal{A}_{(\theta\vec{x})P}$ and $\mathcal{A}_{(\theta\vec{x})Q}$ are f-bisimilar, for $f=\xi((\theta\vec{x})Q)\circ(\xi((\theta\vec{x})P))^{-1}$.*

If the sets of states $\mathcal{S}$ and $\mathcal{S}'$ of the automata are finite, then indexed bisimulations are finite objects.

Theorem 5. *Let $\mathcal{A}_{(\theta\vec{x})P}=(\mathcal{S},\|(\theta\vec{x})P\|,\mapsto)$ and $\mathcal{A}_{(\theta\vec{x})Q}=(\mathcal{S}',\|(\theta\vec{x})Q\|,\mapsto')$, for $P,Q\in\mathcal{P}_{\{x_1,\ldots,x_n\}}$ finitary. Then $P\sim Q$ iff there exists an indexed bisimulation $\mathcal{R}\subseteq\mathcal{S}\times\mathbb{M}(k,k)\times\mathcal{S}'$, with $k=\max(\overline{\deg}(P)\times|n(P)|,\overline{\deg}(Q)\times|n(Q)|)$, such that $(\|(\theta\vec{x})P\|,f,\|(\theta\vec{x})Q\|)\in\mathcal{R}$, where $f=\xi((\theta\vec{x})Q)\circ(\xi((\theta\vec{x})P))^{-1}$.*

Notice that k is an upper bound of the domain of the reindexing functions between all descendents of P and Q. Thus, since $\mathbb{M}(k,k)$ is finite, there are only finitely many candidate relations to be indexed bisimulations. Hence, we have an algorithm for deciding bisimilarity of finitary processes.

6 Final Remarks and Directions for Future Work

In this paper, we have introduced the $\pi\theta$-*calculus,* a conservative extension of the π-calculus, which allows to explain away the mechanism of name creation by means of a new unary binding operator. We have used the $\pi\theta$-calculus to give a coalgebraic description of early bisimilarity. This semantics is finitary, in the sense that it is finitely branching and moreover, for any finitary process, the set of descendants is finite, up-to vacuous θ's and structural congruence. Moreover, the $\pi\theta$-calculus has a direct and clean implementation in Logical Frameworks based on Constructive Type Theory.

Furthermore, we have also introduced θ-*automata* which we use to get a truly finite representation for finitary processes, by equating in a single state a process together with all the processes differing from it by sequences of vacuous θ's and structural congruence. We could push further the study of θ-automata, by introducing a general notion of θ-automaton, independent from the π-calculus. Standard results on automata such as minimalization could then be naturally recovered. Moreover, there should be a corresponding notion of transition system, generalizing the transition system of the $\pi\theta$-calculus.

In [16], an alternative finitary coalgebraic semantics for the π-calculus is proposed for early bisimilarity, and a corresponding notion of automaton, the History Dependent Automaton, is discussed. The problem of generating a fresh name in bound output transitions is solved by applying a suitable permutation on the names of the process, so as to guarantee that a special concrete name is always fresh in the permuted process. This latter name is the new name used in the bound output transition. In a loose sense, this can be viewed as a first-order approach, whereas the one using θ-closure operators of this paper is second-order.

The final coalgebra considered in [16] has the structure of a *permutation algebra*. This allows e.g. to show that the support of the interpretation of a π-calculus process in the final model consists exactly of the active names of the given process. In our presentation, the final coalgebra has the structure of a 1-ary ω-algebra, and the counterpart of the result on the support of [16] is Proposition 8: in the $\frac{1}{\omega}$-algebra of the final model, there are only finitely many different ρ_i-closures of the interpretation of a closed process P, corresponding to the number of different prefix permutations of the active names of P.

In this paper we have considered early bisimilarity, but similar techniques can be used to account for late and weak bisimilarities. Moreover, it would be interesting to explore also denotational (i.e. compositional) models for early/late congruences based on the $\pi\theta$-calculus. We expect to get simpler models than the ones based on functor categories.

Finally, due to the "symbolic" nature of our operational semantics, it would be interesting to compare it to other symbolic approaches that are at the basis of different π-calculus implementations and tools, e.g. [9, 20].

References

1. P. Aczel. Non-well-founded sets. CSLI Lecture Notes 14, Stanford University, 1988.

2. P. Aczel and N. Mendler. A final coalgebra theorem. In *Proc. CTCS'89*, LNCS 389, pages 357–365, Berlin, 1989. Springer-Verlag.
3. F. Borceux. *Handbook of Categorical Algebra, Volume 2*. Number 51 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.
4. M. G. Buscemi and U. Montanari. A first order coalgebraic model of π-calculus early observational equivalence. In *Proc. CONCUR 2002*, LNCS 2421, pages 449–465. Springer, 2002.
5. A. Corradini, R. Heckel, and U. Montanari. Compositional SOS and beyond: A coalgebraic view of open systems. Technical Report TR-01-01, Computer Science Department, University of Pisa, 2001. To appear in *Theoret. Comput. Sci.*
6. J. Despeyroux. A higher-order specification of the π-calculus. In *Proc. of the IFIP TCS'2000*, Sendai, Japan, 2000.
7. M. J. Gabbay. Theory and models of the π-calculus using Fraenkel-Mostowski generalized. Submitted, Aug. 2002.
8. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
9. M. Hennessy and H. Lin. Symbolic bisimulations. *TCS*, 138:353–389, 1995.
10. F. Honsell, M. Lenisa, U. Montanari, and M. Pistore. Final semantics for the π-calculus. In D. Gries, editor, *Proc. PROCOMET'98*. Chapman&Hall, 1998.
11. F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001.
12. INRIA. *The Coq Proof Assistant*, 2002. `http://coq.inria.fr/doc/main.html` .
13. M. Lenisa. *Themes in Final Semantics*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1998.
14. D. Miller and C. Palmidessi. Foundational aspects of syntax. *ACM Computing Surveys (CSUR)*, 31(3es):11, 1999.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Inform. and Comput.*, 100(1):1–77, 1992.
16. U. Montanari and M. Pistore. π-calculus, structured coalgebras, and minimal HD-automata. In *Proc. MFCS 2000*, LNCS 1893, pages 569–578. Springer, 2000.
17. J. Parrow. An introduction to the π-calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.
18. J. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249(1):3–80, 2000.
19. J. Rutten and D. Turi. On the foundations of final semantics: Non-standard sets, metric spaces, partial orders. In *REX Conf. Proc.*, LNCS 666, pages 477–530. Springer-Verlag, 1993.
20. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.
21. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge Universtity Press, 2001.

A Coq code

```
Variable Name,Var : Set.
Inductive Proc: Set :=
    nil     : Proc
  | output  : Name -> Name -> Proc -> Proc
  | input   : Name -> (Name -> Proc) -> Proc
  | tau     : Proc -> Proc
```

```
  | nu      : (Name -> Proc) -> Proc
  | inVar   : Var -> Proc
  | rec     : (Var -> Proc) -> Proc
  | par     : Proc -> Proc -> Proc
  | match   : Name -> Name -> Proc -> Proc.
Coercion inVar : Var >-> Proc.
Inductive TProc : Set :=
    inProc  : Proc -> TProc
  | theta   : (Name -> TProc) -> TProc.
Coercion inProc : Proc >-> TProc.

Inductive FAct : Set :=
    ftau : FAct
  | fout : Name -> Name -> FAct
  | fin  : Name -> Name -> FAct.
Inductive BAct : Set :=
    bout : Name -> BAct
  | bin  : Name -> BAct.
Inductive TAct : Set :=
    inFAct   : FAct -> TAct
  | inBAct   : BAct -> TAct
  | theta_a : (Name -> TAct) -> TAct.
Coercion inFAct : FAct >-> TAct.
Coercion inBAct : BAct >-> TAct.

Inductive trans : TProc -> TAct -> TProc -> Prop :=
   TAU   : (P:Proc)(trans (tau P) ftau P)
 | fIN   : (P:Name->Proc)(x,y:Name)(trans (input x P) (fin x y) (P y))
 | bIN   : (P:Name->Proc)(x:Name)(trans (input x P) (bin x) (theta P))
 | OUT   : (P:Proc)(x,y:Name)(trans (output x y P) (fout x y) P)
 | fPARl : (P,Q,P':Proc)(a:TAct)
           (trans P a P') -> (trans (par P Q) a (par P' Q))
 | bPARl : (P,Q:Proc)(P':Name -> Proc)(a:Name -> TAct)
           (trans P (theta_a a) (theta P')) ->
           (trans (par P Q) (theta_a a) (theta [x:Name](par (P' x) Q)))
 | fPARr : (P,Q,P':Proc)(a:TAct)
           (trans P a P') -> (trans (par Q P) a (par Q P'))
 | bPARr : (P,Q:Proc)(P':Name -> Proc)(a:Name -> TAct)
           (trans P (theta_a a) (theta P')) ->
           (trans (par Q P) (theta_a a) (theta [x:Name](par Q (P' x))))
 | COMl  : (P,P',Q,Q':Proc)(x,y:Name)
           (trans P (fin x y) P') ->
           (trans Q (fout x y) Q') ->
           (trans (par P Q) ftau (par P' Q'))
 | COMr  : (P,P',Q,Q':Proc)(x,y:Name)
```

```
          (trans P (fin x y) P') ->
          (trans Q (fout x y) Q') ->
          (trans (par Q P) ftau (par Q' P'))
| OPEN  : (P,P':Name -> Proc)(x:Name)
          ((y:Name)(trans (P y) (fout x y) (P' y))) ->
          (trans (nu P) (bout x) (theta P'))
| UNFOLD: (P:Var->Proc)(P',Q:Proc)(a:TAct)
          (subst (rec P) P P') ->
          (trans P' a Q) ->
          (trans (rec P) a Q)
| MATCH : (x:Name)(P,Q:Proc)(a:TAct)
          (trans P a Q) -> (trans (match x x P) a Q)
| fRES  : (P,Q:Name->Proc)(a:FAct)
          ((y:Name)(trans (P y) a (Q y))) ->
          (trans (nu P) a (nu Q))
| bRES  : (P:Name->Proc)(P':Name->Name->Proc)(a:BAct)
          ((y:Name)(trans (P y) a (theta [z:Name](P' z y)))) ->
          (trans (nu P) a (theta [z:Name](nu (P' z))))
| CLOSEl: (P,Q:Proc)(P',Q':Name->Proc)(x:Name)
          (trans P (bin x)  (theta P')) ->
          (trans Q (bout x) (theta Q')) ->
          (trans (par P Q) ftau (nu [y:Name](par (P' x) (Q' x))))
| CLOSEr: (P,Q:Proc)(P',Q':Name->Proc)(x:Name)
          (trans P (bin x)  (theta P')) ->
          (trans Q (bout x) (theta Q')) ->
          (trans (par Q P) ftau (nu [y:Name](par (Q' x) (P' x))))
| THETA : (P,Q:Name->TProc)(a:Name->TAct)
          ((x:Name)(trans (P x) (a x) (Q x))) ->
          (trans (theta P) (theta_a a) (theta Q)).
```

B Coalgebras over 1-ary ω-algebras

The aim of this section is to prove that all functors on Alg^1_ω which are *liftings* of a *polynomial* functor on $\mathcal{S}et$ admit *final coalgebra*.

We could give a direct construction using (hyper)set-theoretic tools; however, the categorical setting enlightens the connections with the general treatment of (co)algebraic theories [3, 18].

Definition 11 (Algebraic Functor). *An endofunctor $T : Alg^1_\omega \longrightarrow Alg^1_\omega$ is* algebraic *if respects the algebraic structure, that is, if $f : (A, \rho^A) \to (B, \rho^B)$, then for all $i \in \omega$, we have $Tf \circ \rho^{TA}_i = \rho^{TB}_i \circ Tf$.*

The next step is to relate coalgebras of endofunctors over $\mathcal{S}et$ to coalgebras of endofunctors over Alg^1_ω.

Definition 12. *Let $T : \mathcal{S}et \longrightarrow \mathcal{S}et$ and $V : Alg^1_\omega \longrightarrow \mathcal{S}et$ be two functors. An algebraic functor $T' : Alg^1_\omega \longrightarrow Alg^1_\omega$ is a* lifting *of T along V, if $V \circ T' \cong T \circ V$.*

Lifted functors are particularly important, by virtue of the following results.

Theorem 6. *Let $\mathcal{C}, \mathcal{D}$ be two categories and $F : \mathcal{C} \longrightarrow \mathcal{D}$ be a functor with a left adjoint. Let $T : \mathcal{C} \longrightarrow \mathcal{C}$ and $T' : \mathcal{D} \longrightarrow \mathcal{D}$ be two functors such that there exists a natural isomorphism $\phi : F \circ T \xrightarrow{\sim} T' \circ F$. If $(A, \alpha : A \to TA)$ is a final T-coalgebra, then $(FA, \phi_A \circ F\alpha : FA \to T'(FA))$ is a final T'-coalgebra.*

Proof. The adjoint pair $G \dashv F$ can be lifted to a pair of adjoint functors between the categories of T- and T'- coalgebras. Since any functor with a left adjoint preserves limits and the final object is a limit, then the final object of the former category is preserved in the latter. □

The category of ${}^{1}_{\omega}$-algebras can be seen in the setting of Lawvere's algebraic theories [3, Chap. 3]. In fact, Alg^1_ω corresponds to the category of models over the algebra specification given by the free monoid of ω unary operators:

Proposition 9. $Alg^1_\omega \cong \mathsf{Mod}_{\langle \{\bar{\rho}_i \mid i \in \omega\}^*, \{\overline{\rho\rho'}(x) = \bar{\rho}(\bar{\rho'}(x)), \epsilon x = x\}\rangle}$.

As a consequence, Alg^1_ω enjoys several important properties [3, 3.7.8]:

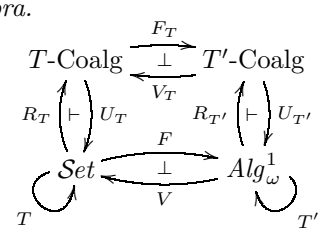
Lemma 4. *The forgetful $V : Alg^1_\omega \longrightarrow Set$ has a left adjoint $F : Set \longrightarrow Alg^1_\omega$.*

Essentially, the left adjoint F is the free algebra construction: the carrier of FA is the set $\{\rho_{i_1} \ldots \rho_{i_n} a \mid a \in A,\ n, i_1, \ldots, i_n \in \omega\}$. By virtue of this adjunction $F \dashv V$ and Theorem 6, we have the following result (see also [5]):

Proposition 10. *Let $T : \mathcal{S}et \longrightarrow \mathcal{S}et$ be a functor, and $T' : Alg^1_\omega \longrightarrow Alg^1_\omega$ be a lifting of T along the forgetful functor $V : Alg^1_\omega \longrightarrow \mathcal{S}et$. Let $U_T : T\text{-Coalg} \longrightarrow \mathcal{S}et$ and $U_{T'} : T'\text{-Coalg} \longrightarrow Alg^1_\omega$ be the obvious forgetful (underlying) functors.*

1. *The forgetful functor $V_T : T'\text{-Coalg} \longrightarrow T\text{-Coalg}$, defined as $V_T(A, \rho^A, \alpha) \triangleq (A, \alpha)$ and $V_T(f) = f$, has a left adjoint $F_T : T\text{-Coalg} \longrightarrow T'\text{-Coalg}$ such that $U_{T'} \circ F_T = F \circ U_T$.*
2. *If U_T has a right adjoint $R_T : \mathcal{S}et \longrightarrow T\text{-Coalg}$, then this lifts to a functor $R_{T'} : Alg^1_\omega \longrightarrow T'\text{-Coalg}$ which is the right adjoint of $U_{T'}$, and such that $R_T \circ V = V_B \circ R_{T'}$.*

Proposition 11. *Let $T : \mathcal{S}et \longrightarrow \mathcal{S}et$ be a functor such that the forgetful $U_T : T\text{-Coalg} \longrightarrow \mathcal{S}et$ has a right adjoint R_T. Let $T' : Alg^1_\omega \longrightarrow Alg^1_\omega$ be a lifting of T along the forgetful $V : Alg^1_\omega \longrightarrow \mathcal{S}et$. Then there exists both the final T-coalgebra and the final T'-coalgebra; moreover, the forgetful $V_T : T'\text{-Coalg} \longrightarrow T\text{-Coalg}$ maps the final T'-coalgebra onto the final T-coalgebra.*



Proof. Right adjoints preserve limits, and final objects are limits. Since $\mathcal{S}et$ has final object $(\mathbf{1})$, the final T-Coalg is $R_T(\mathbf{1})$. By Prop. 10, there exists the right adjoint $R_{T'} : Alg^1_\omega \longrightarrow T'\text{-Coalg}$, which maps the final object $\mathbf{1}$ of Alg^1_ω to the final object of T'-Coalg, that is the final T'-coalgebra. Since $V_T : T'\text{-Coalg} \longrightarrow T\text{-Coalg}$ is a right adjoint, the final object of T'-Coalg is mapped to the final object of T-Coalg, i.e. the final T-coalgebra. □

An important class of functors in $\mathcal{S}et$ is that of *polynomial* functors:

Definition 13. *A functor $F : \mathcal{S}et \longrightarrow \mathcal{S}et$ is* polynomial *if it is definable as a composition of finite products, finite coproducts and finite powerset.*

Proposition 12. *Let $T : \mathcal{S}et \to \mathcal{S}et$ be a polynomial endofunctor. Then, the forgetful functor $U : T\text{-Coalg} \longrightarrow \mathcal{S}et$ has a right adjoint $R : \mathcal{S}et \longrightarrow T\text{-Coalg}$.*

Proof. Following [18], it suffices to prove that $\wp_{\mathrm{f}}(-)$ is bounded, which is true because the subsets assigned by $\wp_{\mathrm{f}}(-)$ are finite. □

Essentially, R is the *coproduct of generators* construction [18, §10].

Finally, we state the main result of this section:

Theorem 7. *Let $T : \mathcal{S}et \longrightarrow \mathcal{S}et$ be a polynomial functor and $T' : Alg^1_\omega \longrightarrow Alg^1_\omega$ be a lifting of T along the forgetful functor $V : Alg^1_\omega \longrightarrow \mathcal{S}et$. Then, there exist both the final T-coalgebra Ω_T and the final T'-coalgebra $\Omega_{T'}$. Moreover, the forgetful functor V_T maps $\Omega_{T'}$ onto Ω_T.*

Proof. Follows from Propositions 12 and 11. We have only to show that the functor on Alg^1_ω by the same polynomial of T is a lifting of T, which is easy by inspection. For instance: $V(\wp_{\mathrm{f}}(A, \rho^A)) = V(\wp_{\mathrm{f}}(A), \rho^{\wp_{\mathrm{f}}(A)}) = \wp_{\mathrm{f}}(A) = \wp_{\mathrm{f}}(V(A, \rho^A))$.
□