Handling Substitutions Explicitely in the π–Calculus

Daniel Hirschkoff

Février 1999

N^o 99-163

Handling Substitutions Explicitely in the π–Calculus

Daniel Hirschkoff

Résumé

Nous reprenons la démarche menant à l'introduction des λ–calculs avec substitutions explicites λs et $\lambda\sigma$ pour présenter πs et $\pi\sigma$, qui se veulent une description des mécanismes mis en œuvre dans le π–calcul pour la manipulation des noms. Alors que πs fait intervenir des opérateurs sur les index de De Bruijn dans la syntaxe des termes, ce qui en fait un bon point de départ afin de comprendre la notation de De Bruijn pour le π–calcul, $\pi\sigma$ est un calcul de noms, termes et substitutions. Cela se traduit en particulier par le fait que les opérations faisant intervenir les substitutions sont introduites sous par l'intermédiaire d'un système de réécriture. Nous établissons une correspondance étroite entre les notions de bisimilarité associées à ces deux calculs, ainsi qu'avec une troisième notion, qui correspond aux termes "usuels" en notation de De Bruijn. Le présent travail donne une description formelle de l'interaction entre les deux opérateurs liants du π–calcul, ce qui permet de mieux comprendre le comportement de l'opérateur de restriction, et peut présenter un intérêt en termes d'implémentation (en particulier dans un assistant à la preuve).

Abstract

We present two calculi aimed at describing the mechanism of name manipulation in the π–calculus. Placing ourselves in the framework of the λs and $\lambda\sigma$ calculi, we define πs and $\pi\sigma$. The former calculus includes operators on De Bruijn indices, and is first introduced to give an intuitive description of the De Bruijn representation of π–calculus terms. The latter is a calculus of explicit substitutions, where the part corresponding to name manipulation is defined as a Term Rewrite System. We introduce the two corresponding notions of bisimulation, and show that they can be put in correspondence; in doing this, we establish a relation with what could be considered "usual" bisimulation on π–calculus terms in De Bruijn notation. These results shed light on the mechanism of name–passing, which can be of interest both for the implementation and the formal treatment (e.g. in a logical framework) of related calculi.

Introduction

Calculi of explicit substitutions originate in Categorical Combinatory Logic, and have been designed for many variants of λ–calculus and related formalisms (e.g. type systems, natural deduction, or higher–order logics). Besides the treatment of (functional) sequential computation, such an approach has also been recently adopted to study the object–oriented paradigm [KL98, LLL98]. Our goal in this paper is to provide a similar account of *concurrency*, through the analysis of one of his most popular algebraic models, namely π–calculus. We concentrate on the mechanism of name–passing, which lies at the heart of the expressiveness of this formalism, and try to understand it by describing its effect on π–calculus terms. In this attempt, our work differs considerably from [FMQ96], where a calculus of explicit substitutions (called $\pi\xi$) is introduced by focusing rather on the *environment* of a π–calculus process during its execution. We shall return on this point later on.

In *name–passing calculi*, the means of communications (typically *channels*, or *names*) can be used as the object of the communication; within such an approach, one can describe systems whose topology is changing along the computation, which can be source of great expressiveness. In the π–calculus, the only information being exchanged between processes is names; this is however enough to encode many paradigms of computer science, thanks to the mechanism of *name extrusion*. Name extrusion can be illustrated on the following example: consider the π–calculus transition

$$a(x).P \mid (\nu c)\,\overline{a}c.Q \xrightarrow{\tau} (\nu c)\,(P_{x:=c} \mid Q)\,.$$

We have here a process liable to *receive* a name x along some channel a, and then to behave like some process P: this term is written $a(x).P$. In the latter expression, P depends in general on x (in the same way a λ–term M depends on x in $\lambda x.\,M$). In parallel with $a(x).P$, we find a process willing to *send* some value (i.e. name) c on a, and then to proceed according to Q. The information that is sent on a is not known to the receiver before the communication: this is represented using the restriction operator ν, that makes the usage of c private to $\overline{a}c.Q$. We thus see that the π–calculus has two binders, namely *abstraction* (embedded in the receiving process, and sometimes written λ) and *restriction* (which, as we shall see, is not directly related to the notion of substitution). As both processes synchronise on a, they perform a τ–transition, which has two effects: in P, the formal parameter x is instanciated by the value being received, namely c; at the same time, the name c is now known by both actors of the synchronisation,

hence the scope of the restriction on c has grown: we observe the extrusion of name c.

Let us now move to the framework of De Bruijn indices [dB72], and see how these mechanisms are implemented. Recall that we are working with two binders, λ and ν ($a(x).P$ and $(\nu c)\,\overline{a}c.Q$ should thus roughly look like "$a\lambda.P$" and "$\nu\overline{a}0.Q$" respectively – notice that we adopt the convention that De Bruijn indices start at 0). From the emitter's point of view, no update on De Bruijn indices needs to be performed, since the execution environment has not been modified; quite remarkably, this holds also for the receiver, because in P all names bound by the abstraction on x become bound by the restriction on c, which means that nothing changes in the De Bruijn representation of P. So the situation seems simpler than in λ–calculus, where β–reduction involves in general some updates in the free names of the term; we recover anyway such a phenomenon in the case of the communication of a free name (i.e. a name not bound by restriction), where actual instanciation takes place. Note however that in the π–calculus, a name is always instanciated with another name, while a whole term can be provided in the λ–calculus. We therefore need to be able to perform two basic manipulations on names, that is replacement of a name with another name, and *name lifting*, to preserve the "meaning" of De Bruijn indices as we cross a binder. In the π–calculus, we need another extra primitive manipulation, which is related to the behaviour of the restriction operator ν. Consider indeed the term $T = (\nu b)(\nu c)\,\overline{a}c.Q$; with respect to the emitting process seen above, we have another private name, b (but only private name c get sent along a). As process T performs the emission of private name c along a, it "sends" the restriction on c, while the restriction on b remains above Q. What happens is that the restriction on c somehow crosses the restriction on b, which will result using the De Bruijn notation in exchanging indices 0 and 1 in Q. This phenomenon leads to treat the ability of permuting two consecutive indices in a term as a primitive action in name manipulation.

According to the considerations we just made, we focus on the following version of π–calculus: the syntax of processes is described by

$$P \;=\; \mathbf{0} \mid \overline{a}b \mid a(b).P \mid (P_1|P_2) \mid (\nu x)\,P\,,$$

where a, b range over a basic sort of *names*. This is a monadic, sum free, finite, asynchronous calculus. Polyadicity will be treated in Section 4, while the other features are not relevant for our study (in particular, the absence of a continuation for emitting processes implies that the examples seen above

cannot be directly treated in our calculus; this is anyway harmless, as the same reasoning could be done within our language). The inactive process, **0**, is written in bold font to distinguish it from the De Bruijn's index 0. We present below the operational semantics of our calculus. Actions, ranged over with μ, can be of three kinds, namely receptions (written $a(b)$), emissions (written $\overline{a}(b)$ or $\overline{a}b$, depending whether the emitted name is private to the sender or not), and synchronisations (written τ). $n(\mu)$, $\mathrm{fn}(\mu)$ and $\mathrm{bn}(\mu)$ denote respectively the sets of names, free and bound names of an action μ (bound names are defined by saying that b is bound in $\overline{a}(b)$). Symmetrical versions of rules PAR_l, COMM_1 and CLOSE_1 are omitted.

$$(\mathrm{INP})\ a(b).P \xrightarrow{a(c)} P_{b:=c} \qquad (\mathrm{OUT})\ \overline{a}b \xrightarrow{\overline{a}b} \mathbf{0}$$

$$(\mathrm{OPEN})\ \frac{P \xrightarrow{\overline{a}b} P'}{(\nu b)\,P \xrightarrow{\overline{a}(\vec{b})} P'}\ b \neq a \qquad (\mathrm{RES})\ \frac{P \xrightarrow{\mu} P'}{(\nu x)\,P \xrightarrow{\mu} (\nu x)\,P'}\ x \notin n(\mu)$$

$$(\mathrm{COMM}_1)\ \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad (\mathrm{PAR}_l)\ \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset$$

$$(\mathrm{CLOSE}_1)\ \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}(b)} Q'}{P|Q \xrightarrow{\tau} (\nu b)\,(P'|Q')}\ b \notin \mathrm{fn}(Q)$$

Following the approach that is followed in the presentation of the λs–calculus [KR95], we introduce πs, the calculus of terms written in the De Bruijn notation. W.r.t. λs, we add to operators for substitution and name lifting an operator ψ, to embed the permutation of two consecutive indices. In λs and $\lambda\sigma$ the process of β reduction is decomposed into a β–rule and some rules to compute the propagation of a substitution in the term; likewise, we introduce here the operational semantics via two relations, written $\xrightarrow{\mu}_s$ and $\rightsquigarrow_s$. $\xrightarrow{\mu}_s$ corresponds to the firing of a transition, that can generate one or several computations on names, described by $\rightsquigarrow_s$. As usual in the De Bruijn framework, some side conditions in the definition of the transition relation can be embedded in the implementation of $\xrightarrow{\mu}_s$, due to the representation of bound names. This presentation naturally gives rise to a notion of bisimilarity, written $\sim_s$, that incorporates both relations. One can then refine the description of the calculus, by adopting the point of view of the $\lambda\sigma$–calculus, and define $\pi\sigma$. Within such an approach, we develop a calculus of processes and substitutions, where the counterpart of operator ψ, written $\updownarrow$, enriches the calculus σ of $\lambda\sigma$. A consequence of this presentation is that the corresponding relation $\rightsquigarrow_\sigma$ is defined as a "plain" Term Rewrite System (without

control structure). One is then interested in relating both formalisms, and we shall see that $\pi\sigma$ can be considered as giving a finer account on the terms described by πs.

As said above, our approach differs from [FMQ96], where the execution of a π–calculus process (with named variables) is described by providing an account of the evolution of its environment, rather than by focusing on the updates to be made *inside* the term, as is the case here. In a certain sense, [FMQ96] is close to an *implementation* of π–calculus terms, in a way that is reminiscent of the abstract machine used to execute PICT programs [Tur95], where typically a ν is viewed as a `new` command, and where its execution results in adding a freshly created name in the heap of names. In the present work, our aim is to focus on the technical details that are hidden in the traditional formal definition of π–calculus terms. Such an approach is in particular of interest for a theorem prover formalisation of π–calculus, where one may want to stay close to the "mathematical" definition rather than design a specific implementation of processes. At larger scale, such an attempt can also be interesting to understand the meaning of a naming operator like ν, that is not specific to the π–calculus.

The plan of the paper is as follows. Sections 1 and 2 introduce the πs–calculus and the $\pi\sigma$–calculus respectively, together with the corresponding notions of bisimilarity. Section 3 is devoted to the comparison of both formalisms. We establish the correspondence between the two descriptions (Theorem 3.19), using in particular an up–to technique for bisimulation [San95]. In Section 4, we comment on two variants of our approach, namely late semantics and polyadicity, and discuss on the study of bisimilarity proofs in our setting. We conclude and discuss related work in Section 5.

1 The Monadic πs–calculus

We present here an account of π–calculus terms in the De Bruijn notation, and try to give an intuition of this implementation; this part of our work is close to [Amb91], where a translation from π–calculus terms with names into the De Bruijn setting is presented. However, our presentation remains closer to the tradition of substitutions calculi (à la λs), by treating the operators on names as part of the calculus, and not at meta level. Such an approach naturally fits to the introduction of $\pi\sigma$ in the next Section.

1.1 Syntax

Terms of the π–calculus are built upon a basic notion of *name*; in the De Bruijn notation, names (sometimes referred to as *channels*) are represented by natural numbers, interpreted as indices. The definition of the πs–calculus follows the approach of the λs calculus [KR95]; accordingly, the syntactic sorts of terms (and of names) include *operators*, that explicitely handle the manipulations to be made on De Bruijn indices.

Definition 1.1 (πs–calculus – syntactic sorts) *The terms of the πs–calculus are defined as follows:*

$$\begin{array}{rcl} a = \mathbb{N} & \textit{name values} & na = a \mid \boldsymbol{op}_s\, na \quad \textit{names} \\ & \boldsymbol{op}_s = \langle na \rangle^i \mid \varphi^i \mid \psi^i & \textit{operators} \\ P = \mathbf{0} \mid \overline{a}b \mid a\lambda.P \mid (P_1|P_2) \mid \nu P \mid \boldsymbol{op}_s\, P & & \textit{processes} \end{array}$$

Let us comment on the above definitions. The notion of *name value* shall be useful in the definition of the semantics, and corresponds to the sort of names where no occurrence of operators is allowed. Let us remark as well that we use here a form of overloading, as operators can be applied both to names and to processes. We find here two operators from λs, namely substitution and lifting (written φ), while operator ψ is specific to the π–calculus; we shall describe and justify the behaviour of these constructs below. Note that we do not have a precise definition of *"π–calculus terms in the De Bruijn notation"*, and instead directly work within the πs–calculus; however, we shall more or less retrieve such a notion in Section 3, where we shall reason on πs processes without occurrences of the operators (i.e. terms that are in some way "fully evaluated" w.r.t. the calculus of operators).

1.2 Early Operational Semantics

1.2.1 Actions

We now turn to the presentation of the behaviour of πs–calculus terms, by defining an *early* operational semantics for processes. The choice of an early version of the semantics strongly influences our description of the operational semantics, since in such a framework one is compelled to know the shape of the environment where the continuation of a comitting term shall be executed. Accordingly, we are led to separate receptions into *bound* and *free* input actions, in a way that is symmetrical to what is usually done with output actions. We shall see in the next subsection that a synchronisation

involving a name which is private to the sender does not change the context of the communicating agents: the sender still "sees" the restriction after the synchronisation, while from the point of view of the receiver, an abstraction has been replaced by a restriction, still yielding the same meaning for the De Bruijn indices. On the contrary, a communication of a free name involves a "true" substitution, and has to be represented differently in our setting.

The syntax for actions is as follows (we describe the reception and the emission of a restricted name using a somewhat peculiar use of the restriction operator ν):

Definition 1.2 (Actions) Actions, *ranged over by* μ, *are defined by the following syntax:*

$$\mu \;=\; a(b) \mid a(\nu) \mid \overline{a}b \mid \overline{a}(\nu) \mid \tau\,.$$

$\mathrm{bn}(\mu)$ *is the number of* bound names *of an action, and is defined as follows:* $\mathrm{bn}(\mu) = 0$ *for actions of the shape* $a(b)$, $\overline{a}b$ *and* τ, $\mathrm{bn}(\mu) = 1$ *otherwise.*

Note that operators do not arise in the definition of actions: this is related to the introduction of name values in Definition 1.1, and will be explained below.

1.2.2 Representation of the Transition Rules

The behaviour of πs terms is described by two relations; the transition relation $P \xrightarrow{\mu}_s P'$ (meaning that process P is liable to perform action μ and become process P') describes the communicating behaviour of terms, and $\rightsquigarrow_s$ corresponds to the calculus of operators, as they are introduced in the syntax. We now comment on the shape of the transition rules as they are implemented in the De Bruijn notation; these rules are summed up on Figure 1. The computation of operators, as defined on Figure 2, shall be justified along the explanations below.

The rules for prefixed processes are INP, INP_b and OUT; they all involve a premise saying that the components of the action are name values. This means that the prefixes involved in a transition should be fully evaluated. Such a condition is indeed needed for the definition of both the operational semantics (where we have to match two symmetric actions in order to infer a synchronisation) and the behavioural equivalence (the notion of simulation also involves a matching between two actions, namely those performed by the processes to be compared). This requirement is reminiscent of the notion

of *weak head normal form* for the λ–calculus, where the head of the term is compelled to be a variable[1].

The definition of rule OUT is straightforward. As said above, the reception of a restricted (new) name consists, from the receiver point of view, in replacing a λ by a ν; this is (partially) embedded in rule INP_b, the ν being provided as rule CLOSE is applied (see below). Rule INP can be seen as a rule for σ–generation, the substitution of b for 0 in the receiving term being the result of the transition: it is reminiscent of the representation of the β–rule in the λs calculus.

Similarly, PAR and RES can be seen as the rules for φ and ψ generation respectively. They are both written using a form of abuse of notation, in order to factorise the presentation; the subscript $\text{bn}(\mu)$ in the occurrence of the operators is interpreted as follows: if $\text{bn}(\mu) = 0$, then the operator reduces to the identity (i.e. we apply no operator), while $\text{bn}(\mu) = 1$ implies that the operator should be taken as it is, without the subscript. Such a notation allows us to avoid describing the various instances of rules RES and PAR, for each kind of action. In order to describe the meaning of operators φ and ψ, we shall now suppose that $\text{bn}(\mu) = 1$. Rule PAR is used to infer a transition for a parallel composition in the case where one process is performing the action (here P) and the other one (Q) is watching. $\text{bn}(\mu) = 1$ means that when the synchronisation will take place, and extra restriction will be put between Q and its "environment"; therefore, in order to keep the same meaning for the indices of free variables in Q, we have to *lift* all indices by 1: this is achieved by operator φ (the superscript 0 corresponds to the depth where the modification is applied in the term – see below). This operator is thus akin to operator φ of the λs calculus; its behaviour is given on Figure 2.

Similarly, rule RES describes the behaviour of a restricted process in the case where the topmost restriction is not involved in the transition. Here again, we make use of a notation to abbreviate our presentation: $\uparrow\mu$ stands for an action where all free names involved in the action are of the form $k+1$, $k \in \mathbb{N}$ (this includes in particular τ actions), the action μ thus representing the same action where every $k+1$ is replaced by k. Intuitively, whenever a process P is liable to perform action $\uparrow\mu$, then νP can perform the same action "hidden" by a restriction: we thus have to decrease the free variables of μ by one. In P', the result of this transition will be that the restriction

[1]Moreover, this analogy can be related to the encodings of the lazy λ–calculus into the π–calculus [Mil92], where the strategy in "attacking" λ–terms by evaluation of their topmost redex is closely mirrored on the way processes evolve.

that is carried by the action (recall that we suppose $\mathrm{bn}(\mu) = 1$) is *on top* of the "immobile" restriction, while it was under it before the transition: we therefore need an operator ψ, to exchange indices 0 and 1 in P' (the evaluation rules of ψ are given on Figure 2).

In the case where the topmost restriction corresponds to the name that is emitted (that thus has to be 0), rule OPEN applies (while RES does not, for the action is not of the shape $\uparrow\mu$): as before, the location of the emission evolves from $a+1$ into a, while the restriction gets "picked up" by the emission. Symmetrically with respect to the INP_b rule, process P' does not need to be modified, as it already "knows" the restriction that takes part in the communication.

We are left with the rules for synchronisation. Since we work with an early semantics, their formulation is easy, because all the work has been done *before* the communication takes place. COMM and CLOSE are straightforwardly introduced to describe synchronisation in the case of free and bound outputs respectively.

$$(\mathrm{INP})\ \frac{a, b \text{ values}}{a\lambda.P \xrightarrow{a(b)} \langle b\rangle^0 P} \qquad (\mathrm{INP}_\nu)\ \frac{a \text{ value}}{a\lambda.P \xrightarrow{a(\nu)} P}$$

$$(\mathrm{OUT})\ \frac{a, b \text{ values}}{\overline{a}b \xrightarrow{\overline{a}b} \mathbf{0}} \qquad (\mathrm{OPEN})\ \frac{P \xrightarrow{\overline{a+1}0} P'}{\nu\, P \xrightarrow{\overline{a}(\nu)} P'}$$

$$(\mathrm{PAR})\ \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P' \mid \varphi^0_{\mathrm{bn}(\mu)} Q} \qquad (\mathrm{RES})\ \frac{P \xrightarrow{\uparrow\mu} P'}{\nu\, P \xrightarrow{\mu} \psi^0_{\mathrm{bn}(\mu)} P'}$$

$$(\mathrm{COMM})\ \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad (\mathrm{CLOSE})\ \frac{P \xrightarrow{a(\nu)} P' \quad Q \xrightarrow{\overline{a}(\nu)} Q'}{P|Q \xrightarrow{\tau} \nu\,(P'|Q')}$$

Figure 1: πs–calculus Operational Semantics: relation $\xrightarrow{\mu}_s$

Figure 1 presents the rules we have just exposed; as said above, rules (INP), (RES), and (PAR) can be seen as generation rules for operators σ, ψ and φ respectively. One can remark that a transition involves at most one σ–generation and zero or several φ and ψ–generations. Relation $\rightsquigarrow_s$ is the smallest relation satisfying the rules of Figure 2. Each operator is indexed by an integer representing the depth at which it is applied, and which gets

incremented each time we cross a binding construct (λ or ν).

σ–destruction	$\langle b\rangle^i\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad a < i \\ b+i & if \quad a = i \\ a-1 & if \quad a > i \end{cases}$
σ–inp transition	$\langle b\rangle^i\, a\lambda.P$	$\rightsquigarrow_s$	$(\langle b\rangle^i\, a)\lambda.(\langle b\rangle^{i+1}\, P)$
σ–res transition	$\langle b\rangle^i\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\langle b\rangle^{i+1} P)$
σ–out transition	$\langle b\rangle^i \overline{a}c$	$\rightsquigarrow_s$	$\overline{\langle b\rangle^i a}(\langle b\rangle^i c)$
σ–— transition	$\langle b\rangle^i (P_1 \mid P_2)$	$\rightsquigarrow_s$	$\langle b\rangle^i P_1 \mid \langle b\rangle^i P_2$
σ–**0** transition	$\langle b\rangle^i \mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$
φ–destruction	$\varphi^i\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad a < i \\ a+1 & if \quad a \geq i \end{cases}$
φ–inp transition	$\varphi^i\, a\lambda.P$	$\rightsquigarrow_s$	$(\varphi^i\, a)\lambda.(\varphi^{i+1}\, P)$
φ–res transition	$\varphi^i\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\varphi^{i+1}\, P)$
φ–out transition	$\varphi^i\, \overline{a}b$	$\rightsquigarrow_s$	$\overline{\varphi^i a}(\varphi^i b)$
φ–— transition	$\varphi^i\, (P_1 \mid P_2)$	$\rightsquigarrow_s$	$\varphi^i\, P_1 \mid \varphi^i\, P_2$
φ–**0** transition	$\varphi^i\, \mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$
ψ–destruction	$\psi^i\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad a < i \\ i+1 & if \quad a = i \\ i & if \quad a = i+1 \\ a & if \quad a > i+1 \end{cases}$
ψ–inp transition	$\psi^i\, a\lambda.P$	$\rightsquigarrow_s$	$(\psi^i a)\lambda.(\psi^{i+1} P)$
ψ–res transition	$\psi^i\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\psi^{i+1} P)$
ψ–out transition	$\psi^i\, \overline{a}b$	$\rightsquigarrow_s$	$\overline{\psi^i a}(\psi^i b)$
ψ–— transition	$\psi^i\, (P_1 \mid P_2)$	$\rightsquigarrow_s$	$\psi^i P_1 \mid \psi^i P_2$
ψ–**0** transition	$\psi^i \mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$

Figure 2: Monadic πs–calculus operators: relation $\rightsquigarrow_s$

1.2.3 Behavioural Equivalence

Given the definition of relations $\xrightarrow{\mu}_s$ and $\rightsquigarrow_s$, we define an equivalence on πs terms as follows:

Definition 1.3 ($\sim_s$) *A relation $\mathcal{R}$ between terms of the πs–calculus is an s–bisimulation iff whenever $P\mathcal{R}Q$ and $P \overset{*}{\rightsquigarrow}_s \xrightarrow{\mu}_s \overset{*}{\rightsquigarrow}_s P'$, there exists Q' s.t.*

$Q \overset{*}{\rightsquigarrow}_s \xrightarrow{\mu}_s \overset{*}{\rightsquigarrow}_s Q'$ *and* $P' \mathcal{R} Q'$*, and the symmetrical condition on transitions of* Q*.* s*–bisimilarity, written* $\sim_s$*, is the greatest* s*–bisimulation.*

Let us comment on this definition. An observable transition of a process corresponds here to the composition of $\overset{*}{\rightsquigarrow}_s$, $\xrightarrow{\mu}_s$, and $\overset{*}{\rightsquigarrow}_s$ again. One could wonder if we can get rid of one of the two $\overset{*}{\rightsquigarrow}_s$. The first one (before the μ transition) is needed to trigger all possible evolvings of P and Q, in order to respect the branching structure described by the notion of bisimulation: indeed, as no $\xrightarrow{\mu}_s$ transition can occur under an operator, we do not want to miss some transitions because some prefixes are not "evaluated" in a process. One solution to get rid of the first $\overset{*}{\rightsquigarrow}_s$ could thus be to consider relations on terms where all topmost prefixes are evaluated (which would correspond to a notion of *weak head normal form*). Alternatively, one may want to avoid the $\overset{*}{\rightsquigarrow}_s$ *after* the μ step; this would amount to adopt a policy in writing our relations, so that terms involved in the relation are reached right after the μ transition. We have kept the definition above in order to allow any kind of definition for the relations between processes, by preserving symmetry; the properties we shall prove in Section 3 can help giving a precise meaning to the considerations we have just made[2].

2 The Monadic $\pi\sigma$–calculus

We introduce here the $\pi\sigma$–calculus, which is a calculus of names, processes and substitutions (as opposed to πs, where operators are "integrated" into a calculus of only processes and names). $\pi\sigma$ provides a definition of the names–handling mechanism as a "plain" Term Rewrite System, while control constructs are used in πs to describe manipulation of names.

2.1 Syntax

The syntax of $\pi\sigma$–calculus terms is given below; we simultaneously define *names* (ranged over by a, b), *processes* (ranged over by P) and *substitutions* (ranged over by s). According to the presentation of $\lambda\sigma$, integers are not primitive anymore, and are instead represented using constants 0 and $\uparrow$:

[2]Note as well that the choices we examined are reminiscent of the various notions of *weak* equivalences or preorders on π–calculus processes, where τ–transitions play the rôle of $\rightsquigarrow_s$ moves, i.e. in some way of "unobservable" actions – see below.

Definition 2.1 ($\pi\sigma$–calculus terms)

$$\begin{array}{rcll} a & = & 0 \mid a[s] & \textit{names} \\ P & = & \mathbf{0} \mid \overline{a}b \mid a\lambda.P \mid (P_1|P_2) \mid \nu P \mid P[s] & \textit{processes} \\ s & = & id \mid a.s \mid \uparrow \mid \updownarrow \mid s \circ s & \textit{substitutions} \end{array}$$

With respect to the calculus of substitutions of [ACCL91], we remark that we have an extra constant, written $\updownarrow$, that will represent the operator ψ of πs. As will be seen later on, this constant can be encoded in σ, the calculus of substitutions without $\updownarrow$. We decide to keep it for the seek of clarity, and because it corresponds to a primitive operation in the definition of the operational semantics. We shall use some conventions to allievate the notation of explicit substitutions:

Notations. (i) Substitutions composition involving $\uparrow$ and $\updownarrow$ will sometimes be noted without the $\circ$ symbol, e.g. $\uparrow\uparrow\updownarrow$.

(ii) The representation of De Bruijn indices is abbreviated using underlined natural numbers, by writing $\underline{1}, \underline{2}, \ldots$. More generally, $\underline{k}$ stands for $0[\underbrace{\uparrow \ldots \uparrow}_{k \text{ times}}]$ (also written $0[\uparrow^k]$).

2.2 Semantics

The transition relation associated to the terms of the $\pi\sigma$–calculus, written $\rightarrow_\sigma$, is directly adapted from $\rightarrow_s$. *Values* for names correspond to De Bruijn indices representants (written as underlined integers), and actions are introduced accordingly, exactly like in πs. $\rightarrow_\sigma$ is defined on Figure 3.

The corresponding calculus of substitutions, given by relation $\rightsquigarrow_\sigma$, is defined on Figure 4. The rewrite rules can be decomposed into three sets; a first set of rules, involving all the substitutions constructors except $\updownarrow$, comes from the calculus σ of [ACCL91]. Rules 0–$\updownarrow$ to $\uparrow$–$\uparrow$–$\updownarrow$–s deal with constant $\updownarrow$, while the rules at the bottom of the Figure propagate substitutions inside terms.

Definition 2.2 ($\sim_\sigma$) *σ–bisimulation and $\sim_\sigma$ are defined as in Definition 1.3, where $\xrightarrow{\mu}_\sigma$ and $\rightsquigarrow_\sigma$ replace $\xrightarrow{\mu}_s$ and $\rightsquigarrow_s$ respectively.*

3 Properties of πs and $\pi\sigma$

We now turn to the comparison between πs and $\pi\sigma$, and the associated notions of bisimilarity. As will be seen, it turns out that these calculi basically describe the same behaviours, and differ only in the granularity of

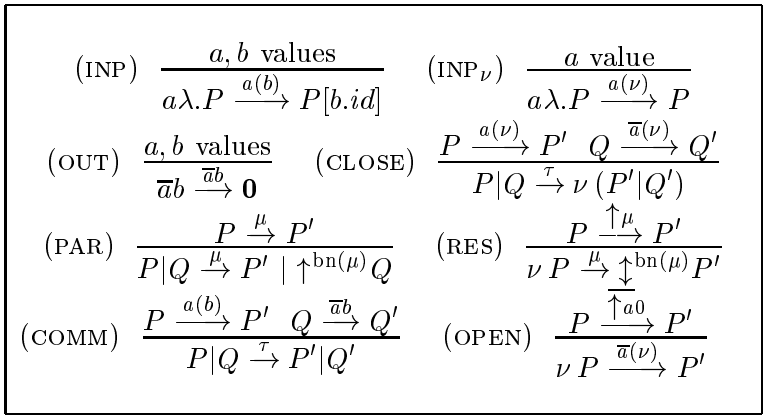


Figure 3: $\pi\sigma$–calculus transition relation

the description ($\pi\sigma$ being finer than πs). A third notion of processes, corresponding to π–calculus terms in De Bruijn notation, will arise along our study.

3.1 Confluence Properties of the Substitutions Calculi

We first focus on the calculi of substitutions, given by relations $\rightsquigarrow_s$ and $\rightsquigarrow_\sigma$; we prove uniqueness of normal forms for relation $\rightsquigarrow_\sigma$, and then define a translation from πs to $\pi\sigma$ that will allow to establish the same result for $\rightsquigarrow_s$.

Proposition 3.1 *$\rightsquigarrow_\sigma$ is strongly normalising.*

Proof. *We exploit the strong normalisation of σ, the subpart of $\lambda\sigma$ [ACCL91] that handles substitutions, by encoding the calculus of substitutions of $\pi\sigma$ into a $\updownarrow$–free calculus, and encoding the resulting calculus of processes into $\lambda\sigma$.*

We define a function $\mathcal{U}_{\pi\to\lambda}$, from $\pi\sigma$ to $\lambda\sigma$, on Figure 5; it is decomposed into three functions, namely $\mathcal{U}^n_{\pi\to\lambda}$, $\mathcal{U}^p_{\pi\to\lambda}$ and $\mathcal{U}^s_{\pi\to\lambda}$, to compute the image of respectively a name, a process, and a substitution (notice that De Bruijn's index 0 corresponds to 1 in $\lambda\sigma$). We easily verify that the definition of $\mathcal{U}_{\pi\to\lambda}$ is well–typed. Moreover, we have the following property:

Lemma 3.2 *$P \rightsquigarrow_\sigma Q$ implies $\mathcal{U}_{\pi\to\lambda}(P) \xrightarrow{+}_\sigma \mathcal{U}_{\pi\to\lambda}(Q)$.*

0–id	$0[id]$	$\rightsquigarrow_\sigma$	0
0–cons	$0[a.s]$	$\rightsquigarrow_\sigma$	a
[]–[]	$a[s][t]$	$\rightsquigarrow_\sigma$	$a[s \circ t]$
id–s	$id \circ s$	$\rightsquigarrow_\sigma$	s
↑–id	$\uparrow \circ id$	$\rightsquigarrow_\sigma$	$\uparrow$
↑–cons	$\uparrow \circ (a.s)$	$\rightsquigarrow_\sigma$	s
cons–∘	$(a.s) \circ t$	$\rightsquigarrow_\sigma$	$a[t].(s \circ t)$
∘–∘	$(s_1 \circ s_2) \circ s_3$	$\rightsquigarrow_\sigma$	$s_1 \circ (s_2 \circ s_3)$
0–↕	$0[\updownarrow]$	$\rightsquigarrow_\sigma$	$\underline{1}$
1–↕	$\underline{1}[\updownarrow]$	$\rightsquigarrow_\sigma$	0
0–↕–s	$0[\updownarrow \circ s]$	$\rightsquigarrow_\sigma$	$\underline{1}[s]$
1–↕–s	$\underline{1}[\updownarrow \circ s]$	$\rightsquigarrow_\sigma$	$0[s]$
↑–↑–↕	$\uparrow \circ (\uparrow \circ \updownarrow)$	$\rightsquigarrow_\sigma$	$\uparrow \circ \uparrow$
↑–↑–↕–s	$\uparrow \circ (\uparrow \circ (\updownarrow \circ s))$	$\rightsquigarrow_\sigma$	$\uparrow \circ (\uparrow \circ s)$
s–inp	$(a.\lambda P)[s]$	$\rightsquigarrow_\sigma$	$\underline{a[s]}.\lambda(P[0.s \circ \uparrow])$
s–out	$(\overline{a}b)s$	$\rightsquigarrow_\sigma$	$\overline{a[s]}b[s]$
s–ν	$(\nu P)[s]$	$\rightsquigarrow_\sigma$	$\nu(P[s])$
s–par	$(P_1 \mid P_2)[s]$	$\rightsquigarrow_\sigma$	$P_1[s] \mid P_2[s]$
s–**0**	$\mathbf{0}[s]$	$\rightsquigarrow_\sigma$	$\mathbf{0}$

Figure 4: Monadic $\pi\sigma$–calculus: calculus of substitutions

We then proceed by contradiction: if there is an infinite $\rightsquigarrow_\sigma$–derivation starting from P, then there exists an infinite $\rightarrow_\sigma$ derivation starting from $\mathcal{U}_{\pi\to\lambda}(P)$, which is impossible by [ACCL91]: this concludes the proof. ♢

Proposition 3.3 *$\rightsquigarrow_\sigma$ is locally confluent.*

Proof. *By critical pairs inspection.* ♢

Newmann's lemma guarantees, using Propositions 3.1 and 3.3, the uniqueness of normal forms for $\rightsquigarrow_\sigma$. Let us now turn to $\rightsquigarrow_s$; we exploit the results we just proved through a translation from πs into $\pi\sigma$. We first need some notation.

$$
\begin{array}{rcll}
\mathcal{U}^n_{\pi\to\lambda}(0) & = & 1 & \text{names} \\
\mathcal{U}^n_{\pi\to\lambda}(a[s]) & = & \mathcal{U}^n_{\pi\to\lambda}(a)[\mathcal{U}^s_{\pi\to\lambda}(s)] & \\
\mathcal{U}^p_{\pi\to\lambda}(\mathbf{0}) & = & 1 & \text{processes} \\
\mathcal{U}^p_{\pi\to\lambda}(a\lambda.P) & = & (\mathcal{U}^n_{\pi\to\lambda}(a)\,\lambda\mathcal{U}^p_{\pi\to\lambda}(P)) & \\
\mathcal{U}^p_{\pi\to\lambda}(\overline{a}b) & = & (\mathcal{U}^n_{\pi\to\lambda}(a)\,\mathcal{U}^n_{\pi\to\lambda}(b)) & \\
\mathcal{U}^p_{\pi\to\lambda}(P_1|P_2) & = & (\mathcal{U}^p_{\pi\to\lambda}(P_1)\;\mathcal{U}^p_{\pi\to\lambda}(P_2)) & \\
\mathcal{U}^p_{\pi\to\lambda}(\nu P) & = & \lambda\mathcal{U}^p_{\pi\to\lambda}(P) & \\
\mathcal{U}^p_{\pi\to\lambda}(P[s]) & = & \mathcal{U}^p_{\pi\to\lambda}(P)[\mathcal{U}^s_{\pi\to\lambda}(s)] & \\
\mathcal{U}^s_{\pi\to\lambda}(id) & = & id & \text{substitutions} \\
\mathcal{U}^s_{\pi\to\lambda}(\uparrow) & = & \uparrow & \\
\mathcal{U}^s_{\pi\to\lambda}(a.s) & = & \mathcal{U}^n_{\pi\to\lambda}(a).\mathcal{U}^s_{\pi\to\lambda}(s) & \\
\mathcal{U}^s_{\pi\to\lambda}(s_1\circ s_2) & = & \mathcal{U}^s_{\pi\to\lambda}(s_1)\circ\mathcal{U}^s_{\pi\to\lambda}(s_2) & \\
\mathcal{U}^s_{\pi\to\lambda}(\updownarrow) & = & [(1[\uparrow]).1.\uparrow\uparrow] &
\end{array}
$$

Figure 5: Function $\mathcal{U}_{\pi\to\lambda}$

Abbreviations. We define the following substitutions:

$$
id^i(s) \stackrel{def}{=} 0.\underline{1}.\ldots.\underline{i-1}.s \qquad a^i \stackrel{def}{=} id^i(a[\uparrow^i].\uparrow^{i+1})
$$

$$
\underset{i}{\uparrow} \stackrel{def}{=} id^i(\uparrow^{i+1}) \qquad \underset{i}{\updownarrow} \stackrel{def}{=} id^i(\updownarrow\circ\uparrow^i)
$$

Definition 3.4 (Translation from πs–calculus to $\pi\sigma$–calculus) *The translation function $\mathcal{T}$ from πs to $\pi\sigma$ is defined on Figure 6. Note that we adopt an overloaded notation, so that $\mathcal{T}$ acts on names, processes and operators). Moreover, the translation of names and actions being straightforward, we shall not mention applications of $\mathcal{T}$ on such constructs, and write names and actions in the same way in πs and $\pi\sigma$.*

We remark that $\mathcal{T}$ is injective, which will be useful below. As expected, $\mathcal{T}$ allows us to reflect πs–transitions into $\pi\sigma$–transitions:

Lemma 3.5 *(i) $P\rightsquigarrow_s P'$ implies $\mathcal{T}(P)\stackrel{+}{\rightsquigarrow}_\sigma\mathcal{T}(P')$*
(ii) $P\stackrel{\mu}{\longrightarrow}_s P'$ iff $\mathcal{T}(P)\stackrel{\mu}{\longrightarrow}_\sigma\mathcal{T}(P')$.

This Lemma indicates that the evolution of a πs–process can be simulated by its translation in $\pi\sigma$; note that *(ii)* basically boils down to saying

$$
\begin{array}{rcll}
k \in \mathbb{N}, \mathcal{T}(k) & = & \underline{k} & \text{names} \\
\mathcal{T}(\mathbf{op}_s\, na) & = & \mathcal{T}(na)[\mathcal{T}(\mathbf{op}_s)] & \\
\mathcal{T}(\mathbf{0}) & = & \mathbf{0} & \text{processes} \\
\mathcal{T}(a\lambda.P) & = & \mathcal{T}(a)\lambda.\mathcal{T}(P) & \\
\mathcal{T}(\overline{a}b) & = & \overline{\mathcal{T}(a)}\mathcal{T}(b) & \\
\mathcal{T}(P_1|P_2) & = & \mathcal{T}(P_1) \mid \mathcal{T}(P_2) & \\
\mathcal{T}(\nu P) & = & \nu\mathcal{T}(P) & \\
\mathcal{T}(\mathbf{op}_s\, P) & = & \mathcal{T}(P)[\mathcal{T}(\mathbf{op}_s)] & \\
\mathcal{T}(\phi^i) & = & \uparrow_i & \text{operators} \\
\mathcal{T}(\psi^i) & = & \updownarrow_i & \\
\mathcal{T}(\langle a \rangle^i) & = & a^i.id &
\end{array}
$$

Figure 6: Function $\mathcal{T}$

that the translation of an evaluated prefix is itself an evaluated prefix. We easily get strong normalisation for $\rightsquigarrow_s$:

Proposition 3.6 *$\rightsquigarrow_s$ is strongly normalising.*

Proof. *We proceed as in the proof of Proposition 3.1, and exploit the strong normalisation of $\rightsquigarrow_\sigma$ by encoding a $\rightsquigarrow_s$–transition into several (in general) $\rightsquigarrow_\sigma$–transitions, using Lemma 3.5.* ♢

We can thus talk about normal forms for $\rightsquigarrow_s$; they are described using the following Lemma:

Lemma 3.7 (Description of $\rightsquigarrow_s$–normal forms) *Every πs–term reduces to a term described by the following syntax:*

$$a = \mathbb{N}, \qquad P = \mathbf{0} \mid \overline{a}b \mid a\lambda.P \mid (P_1|P_2) \mid \nu P\,.$$

Clearly, a process P that obeys this syntax cannot be rewritten using $\rightsquigarrow_s$; this also holds for $\mathcal{T}(P)$, using $\rightsquigarrow_\sigma$.

To establish the confluence of $\rightsquigarrow_s$, we need some results about processes that cannot evolve using $\rightsquigarrow_\sigma$.

Notation. We write $P \rightsquigarrow_\sigma$ to mean that there exists P' s.t. $P \rightsquigarrow_\sigma P'$; when this does not hold, we write $P \not\rightsquigarrow_\sigma$ (we extend this notation to $\not\rightsquigarrow_s$). For any process P of the $\pi\sigma$–calculus, we define $P{\downarrow_\sigma}$ as the *unique* process P_0 such that $P \stackrel{*}{\rightsquigarrow}_\sigma P_0$ and $P_0 \not\rightsquigarrow_\sigma$. Accordingly, we write $P \stackrel{*}{\rightsquigarrow}_\sigma P'{\downarrow_\sigma}$ to denote the computation of a normal form for σ.

Lemma 3.8 $(P \not\rightsquigarrow_s) \Leftrightarrow (\mathcal{T}(P) \not\rightsquigarrow_\sigma)$.

Proof. $\underline{\Leftarrow}$: *by Lemma 3.5.* $\underline{\Rightarrow}$: *by Lemma 3.7.* ♢

Proposition 3.9 *$\rightsquigarrow_s$ is locally confluent.*

Proof. *Suppose that we have a πs–process P and two terms P_1 and P_2 s.t. $P \rightsquigarrow_s P_1$, $P \rightsquigarrow_s P_2$. By nœtherianity of $\rightsquigarrow_s$, we can compute a term Q_1 s.t. $P_1 \stackrel{*}{\rightsquigarrow}_s Q_1$ and $Q_1 \not\rightsquigarrow_s$, and similarly Q_2 from P_2. Suppose then that we have $Q_1 \neq Q_2$; by Lemma 3.8 and injectivity of $\mathcal{T}$, we have that $\mathcal{T}(Q_1) \not\rightsquigarrow_\sigma$, $\mathcal{T}(Q_2) \not\rightsquigarrow_\sigma$, and $\mathcal{T}(Q_1) \neq \mathcal{T}(Q_2)$. But, using Lemma 3.5, we can reconstruct the computation paths from $\mathcal{T}(P)$ to $\mathcal{T}(Q_1)$ and $\mathcal{T}(Q_2)$: this contradicts the uniqueness of normal forms for $\rightsquigarrow_\sigma$. We thus have $Q_1 = Q_2$, and $\rightsquigarrow_s$ is locally confluent.* ♢

We thus have uniqueness of normal forms for $\rightsquigarrow_s$; this allows us to introduce the notation $P{\downarrow_s}$.

Lemma 3.10 *$\mathcal{T}(P{\downarrow_s}) = \mathcal{T}(P){\downarrow_\sigma}$. Reciprocally, $P \not\rightsquigarrow_\sigma$ implies $\exists P_0 . P = \mathcal{T}(P_0) \wedge P_0 \not\downarrow_s$.*

3.2 Relating Behavioural Equivalences

We now turn to the comparison between $\sim_s$ and $\sim_\sigma$. For this task, we first prove some results that allow us to reason on terms which cannot be rewritten using $\rightsquigarrow_\sigma$. Similar properties for $\rightsquigarrow_s$ will then make it possible to establish a close correspondence between $\sim_s$ and $\sim_\sigma$. We first introduce an useful up–to technique [San95]; to do this, we need to show that relations $\rightsquigarrow_\sigma$ and $\stackrel{\mu}{\rightarrow}_\sigma$ commute, which intuitively is guaranteed by the requirement on prefixes to be evaluated in order to fire a $\stackrel{\mu}{\rightarrow}_\sigma$–transition:

Lemma 3.11 *Suppose $P \stackrel{\mu}{\rightarrow}_\sigma Q$ and $P \stackrel{*}{\rightsquigarrow}_\sigma P'$; then there exists Q' s.t. $Q \stackrel{*}{\rightsquigarrow}_\sigma Q'$ and $Q \stackrel{\mu}{\rightarrow}_\sigma Q'$.*

Theorem 3.12 (σ–bisimulation up to σ proof technique) *We say that a relation $\mathcal{R}$* progresses to *a relation $\mathcal{S}$, written $\mathcal{R} \to \mathcal{S}$, iff whenever $P\,\mathcal{R}\,Q$ and $P \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'$, there exists Q' s.t. $Q \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q'$ and $P'\,\mathcal{S}\,Q'$, and the symmetrical condition on transitions of Q.*

Given a relation $\mathcal{R}$, we define $\mathcal{F}_\sigma(\mathcal{R})$ as follows:

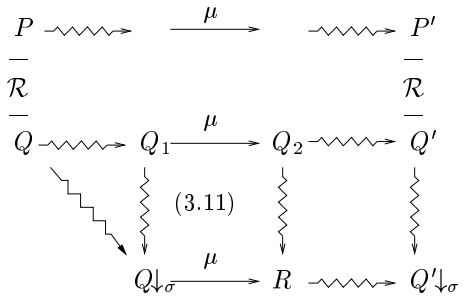
$$\mathcal{F}_\sigma(\mathcal{R}) \;=\; \mathcal{R} \cup \{(P, Q{\downarrow_\sigma}).\, P\,\mathcal{R}\,Q\} \cup \{(P{\downarrow_\sigma}, Q).\, P\,\mathcal{R}\,Q\}\,.$$

Then, for any relation $\mathcal{R}$, $\mathcal{R} \to \mathcal{F}_\sigma(\mathcal{R})$ implies $\mathcal{R} \subseteq \sim_\sigma$.

Proof. *We apply the theory of progressions of relations of [San95], and prove that $\mathcal{F}_\sigma$ is* respectful, *i.e. that ($\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \to_\sigma \mathcal{S}$) implies ($\mathcal{F}_\sigma(\mathcal{R}) \subseteq \mathcal{F}_\sigma(\mathcal{S})$ and $\mathcal{F}_\sigma(\mathcal{R}) \to_\sigma \mathcal{F}_\sigma(\mathcal{S})$). This is indeed sufficient to obtain the desired result. The first property is immediate. For the second one, suppose $\mathcal{R} \to_\sigma \mathcal{S}$, and take $(A, B) \in \mathcal{F}_\sigma(\mathcal{R})$. We only treat the case $(A, B) = (P, Q{\downarrow_\sigma})$, the case $A\,\mathcal{R}\,B$ being trivial and the other case being fully symmetrical.*

Suppose then $Q{\downarrow_\sigma} \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q'$: this is the easy, case, since this means that $Q \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma} \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q'$, and by hypothesis since $P\,\mathcal{R}\,Q$, there exists P' s.t. $P \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'$ and $P'\,\mathcal{S}\,Q'$. Then we have indeed $P'\,\mathcal{F}_\sigma(\mathcal{S})\,Q'$.

The interesting case is when $P \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\longrightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'$. By hypothesis, we can exhibit Q_1, Q_2 and Q' s.t. $Q \stackrel{*}{\rightsquigarrow}_\sigma Q_1 \stackrel{\mu}{\longrightarrow}_\sigma Q_2 \stackrel{*}{\rightsquigarrow}_\sigma Q'$ and $P'\,\mathcal{S}\,Q'$. By confluence of $\rightsquigarrow_\sigma$, since $Q \stackrel{*}{\rightsquigarrow}_\sigma Q_1$, $Q_1 \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma}$; then, since $Q_1 \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma}$ and $Q_1 \stackrel{\mu}{\longrightarrow}_\sigma Q_2$, by Lemma 3.11, we can exhibit a process T s.t. $Q{\downarrow_\sigma} \stackrel{\mu}{\longrightarrow}_\sigma R$ and $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma R$. We now have $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma Q'$ and $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma R$: still by confluence of $\rightsquigarrow_\sigma$, they both reduce by $\stackrel{*}{\rightsquigarrow}_\sigma$ into $Q'{\downarrow_\sigma}$. Finally, we have $Q{\downarrow_\sigma} \stackrel{\mu}{\longrightarrow}_\sigma R \stackrel{*}{\rightsquigarrow}_\sigma Q'{\downarrow_\sigma}$, and since $P'\,\mathcal{S}\,Q'$, $P'\,\mathcal{F}_\sigma(\mathcal{S})\,Q'{\downarrow_\sigma}$, which concludes the proof. Here is a schema to illustrate the reasoning we make:

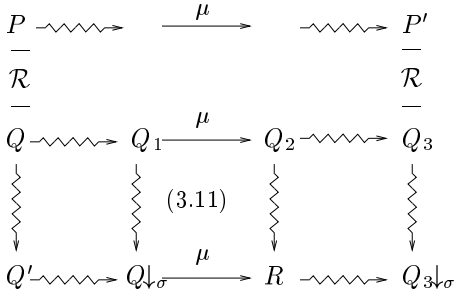


♢

We can now establish the following property, that is also in some sense a proof technique, needed to obtain our first main result (Theorem 3.14):

Proposition 3.13 *Suppose we have two $\pi\sigma$–processes Q and Q' s.t. $Q \stackrel{*}{\rightsquigarrow}_\sigma Q'$; then $(P \sim_\sigma Q)$ iff $(P \sim_\sigma Q')$.*

Proof. $\underline{\Rightarrow}$: *let $\mathcal{R}$ be a σ–bisimulation s.t. $P \mathcal{R} Q$. We show that $\mathcal{R}' = \mathcal{R} \cup \{(P, Q')\}$ is a σ–bisimulation up to σ; the only interesting case is given by the pair (P, Q'). Suppose then $P \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'$, then by hypothesis there exist Q_1, Q_2 and Q_3 s.t. $Q \stackrel{*}{\rightsquigarrow}_\sigma Q_1 \stackrel{\mu}{\rightarrow}_\sigma Q_2 \stackrel{*}{\rightsquigarrow}_\sigma Q_3$ and $P' \mathcal{R} Q_3$. Since $Q \stackrel{*}{\rightsquigarrow}_\sigma Q_1$ and $Q \stackrel{*}{\rightsquigarrow}_\sigma Q'$, by confluence we get $Q' \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma}$ and $Q_1 \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma}$. Now, as $Q_1 \stackrel{\mu}{\rightarrow}_\sigma Q_2$ and $Q_1 \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma}$, by Lemma 3.11 we can exhibit R s.t. $Q{\downarrow_\sigma} \stackrel{\mu}{\rightarrow}_\sigma R$ and $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma R$. We finally use the confluence of $\stackrel{*}{\rightsquigarrow}_\sigma$ to show that, since $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma R$ and $Q_2 \stackrel{*}{\rightsquigarrow}_\sigma Q_3$, $R \stackrel{*}{\rightsquigarrow}_\sigma Q_3{\downarrow_\sigma}$. We thus have $Q' \stackrel{*}{\rightsquigarrow}_\sigma Q{\downarrow_\sigma} \stackrel{\mu}{\rightarrow}_\sigma R \stackrel{*}{\rightsquigarrow}_\sigma Q_3{\downarrow_\sigma}$, and $P' \mathcal{F}_\sigma(\mathcal{R}') Q_3{\downarrow_\sigma}$ (because $P' \mathcal{R} Q_3$).*



The case where $Q' \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q''$ is much easier, since this means that $Q \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q''$, and by hypothesis we can exhibit P' s.t. $P \stackrel{*}{\rightsquigarrow}_\sigma \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'$ and $P' \mathcal{R} Q''$, thus $P' \mathcal{F}_\sigma(\mathcal{R}') Q''$.

$\underline{\Leftarrow}$: *we now have a σ–bisimulation $\mathcal{R}$ s.t. $P \mathcal{R} Q'$. We take $\mathcal{R}' = \mathcal{R} \cup \{(P, Q)\}$, and prove that $\mathcal{R}'$ is a σ–bisimulation. The proof is as in the case above.* ♢

Theorem 3.14 *$P \sim_\sigma Q$ iff $P{\downarrow_\sigma} \sim_\sigma Q{\downarrow_\sigma}$.*

Proof. *Use Proposition 3.13 and the symmetry of $\sim_\sigma$.* ♢

Manipulating processes of the form $P{\downarrow_\sigma}$ suggests another notion of bisimulation, relating only such terms:

Definition 3.15 ($\sim_{\sigma\downarrow}$) *A relation $\mathcal{R}$ between $\pi\sigma$–processes is a $\sim_{\sigma\downarrow}$–bisimulation iff for all $(P, Q) \in \mathcal{R}$, $P \not\rightsquigarrow_\sigma$, $Q \not\rightsquigarrow_\sigma$, and, whenever $P \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma P'{\downarrow_\sigma}$, there exists Q' s.t. $Q \stackrel{\mu}{\rightarrow}_\sigma \stackrel{*}{\rightsquigarrow}_\sigma Q'{\downarrow_\sigma}$ and $P'{\downarrow_\sigma} \mathcal{R} Q'{\downarrow_\sigma}$, and the symmetrical condition on transitions of Q. $\sim_{\sigma\downarrow}$ is the greatest $\sim_{\sigma\downarrow}$–bisimulation.*

Lemma 3.16 *$P\downarrow_\sigma \sim_\sigma Q\downarrow_\sigma$ iff $P\downarrow_\sigma \sim_{\sigma\downarrow} Q\downarrow_\sigma$.*

Proof. $\underline{\Rightarrow}$: *let $\mathcal{R}$ be a $\sim_\sigma$–bisimulation s.t. $P\,\mathcal{R}\,Q$. Define $\mathcal{R}' = \{(X\downarrow_\sigma, Y\downarrow_\sigma).\, X\,\mathcal{R}\,Y\}$. We obviously have $P\downarrow_\sigma \mathcal{R}'\, Q\downarrow_\sigma$; let us prove that $\mathcal{R}$' is a $\sim_{\sigma\downarrow}$–bisimulation.*

Suppose then $X\downarrow_\sigma \mathcal{R}'\, Y\downarrow_\sigma$ and $X\downarrow_\sigma \xrightarrow{\mu}_\sigma \overset{*}{\rightsquigarrow}_\sigma X'\downarrow_\sigma$; then by hypothesis, there exists Y' s.t. $Y \xrightarrow{\mu}_\sigma \overset{*}{\rightsquigarrow}_\sigma Y'$ and $X'\,\mathcal{R}\,Y'$. This shows that $Y \xrightarrow{\mu}_\sigma \overset{*}{\rightsquigarrow}_\sigma Y'\downarrow_\sigma$ and $X'\,\mathcal{R}'\,Y'\downarrow_\sigma$. The symmetric case is treated in an identical fashion.

$\underline{\Leftarrow}$: *$\mathcal{R}$ is now a $\sim_{\sigma\downarrow}$–bisimulation s.t. $P\downarrow_\sigma \mathcal{R}\, Q\downarrow_\sigma$, consider*

$$\mathcal{R}' \;=\; \{(X,Y).\, X\,\mathcal{R}\,Y\downarrow_\sigma \vee X\downarrow_\sigma \mathcal{R}\,Y\}\,.$$

To prove that $\mathcal{R}$' is a $\sim_\sigma$–bisimulation, we consider a transition $X \overset{*}{\rightsquigarrow}_\sigma X_1 \xrightarrow{\mu}_\sigma X_2 \overset{*}{\rightsquigarrow}_\sigma X'$; by confluence of $\rightsquigarrow_\sigma$, $X_1 \overset{*}{\rightsquigarrow}_\sigma X\downarrow_\sigma$, and hence using Lemma 3.11, we can exhibit T s.t. $X\downarrow_\sigma \xrightarrow{\mu}_\sigma T$ and $X_2 \overset{*}{\rightsquigarrow}_\sigma T$. We use again the confluence of $\rightsquigarrow_\sigma$ to show that $T \overset{*}{\rightsquigarrow}_\sigma X'\downarrow_\sigma$, and we thus have $X\downarrow_\sigma \xrightarrow{\mu}_\sigma T \overset{*}{\rightsquigarrow}_{\sigma\downarrow} X'\downarrow_\sigma$. Using the hypothesis that $\mathcal{R}$ is a $\sim_{\sigma\downarrow}$–bisimulation, we can exhibit $Y'\downarrow_\sigma$ s.t. $Y\downarrow_\sigma \xrightarrow{\mu}_\sigma \overset{*}{\rightsquigarrow}_\sigma Y'\downarrow_\sigma$ and $X'\downarrow_\sigma \mathcal{R}\, Y'\downarrow_\sigma$. This is enough, because we have $Y \overset{*}{\rightsquigarrow}_\sigma Y\downarrow_\sigma \xrightarrow{\mu}_\sigma \overset{*}{\rightsquigarrow}_\sigma Y'\downarrow_\sigma$, and $X'\,\mathcal{R}'\,Y'\downarrow_\sigma$, since $X'\downarrow_\sigma \mathcal{R}\, Y'\downarrow_\sigma$. ♢

Lemma 3.16 is interesting because $\sim_{\sigma\downarrow}$–bisimulations are in general "much smaller" than full $\sim_\sigma$–bisimulations, as the proof above suggests. Faithfully following the reasoning above[3], we establish corresponding results for $\sim_s$:

Theorem 3.17 *(i) $P\sim_s Q$ iff $P\downarrow_s \sim_s Q\downarrow_s$ and (ii) $P\downarrow_s \sim_s Q\downarrow_s$ iff $P\downarrow_s \sim_{s\downarrow} Q\downarrow_s$.*

Intuitively, the notions of $\sim_{s\downarrow}$– and $\sim_{\sigma\downarrow}$–bisimulation correspond to bisimulation on "usual" π–calculus terms in the De Bruijn notation: a transition of such a term corresponds to a transition in one of these calculi followed by the full evaluation according to the corresponding calculus of substitutions. We shall now make such a correspondence more precise, by relating $\sim_{s\downarrow}$ and $\sim_{\sigma\downarrow}$, using $\mathcal{T}$.

Corollary 3.18 (of Lemma 3.5) *$P\downarrow_s \xrightarrow{\mu}_s P'$ iff $\mathcal{T}(P\downarrow_s) \xrightarrow{\mu}_\sigma \mathcal{T}(P')$.*

Theorem 3.19 *$P\sim_s Q$ iff $\mathcal{T}(P)\sim_\sigma \mathcal{T}(Q)$.*

Proof. *Using Theorems 3.14 and 3.17 and Lemma 3.16, we show that $P\downarrow_s \sim_{s\downarrow} Q\downarrow_s$ iff $\mathcal{T}(P\downarrow_s)\sim_{\sigma\downarrow} \mathcal{T}(Q\downarrow_s)$.*

[3] In particular, due to the similar definitions of $\sim_s$ and $\sim_\sigma$, the counterpart of Lemma 3.11 is proved in the same way.

$\underline{\Rightarrow}$: *let* $\mathcal{R}$ *be a* $\sim_{s\downarrow}$*–bisimulation, we show that*

$$\mathcal{T}(\mathcal{R}) \;=\; \{(\mathcal{T}(P),\mathcal{T}(Q)).\, P\,\mathcal{R}\,Q\}$$

is a $\sim_{\sigma\downarrow}$*–bisimulation. Indeed, whenever we have* $\mathcal{T}(P)\,\mathcal{T}(\mathcal{R})\,\mathcal{T}(Q)$ *and* $\mathcal{T}(P)\xrightarrow{\mu}_{\sigma}\overset{*}{\rightsquigarrow}_{\sigma}P'{\downarrow_\sigma}$*, we can use results 3.18 and 3.10 to show that* $P\xrightarrow{\mu}_{s}\overset{*}{\rightsquigarrow}_{s}P_0{\downarrow_\sigma}$ *and that* $\mathcal{T}(P_0)=P'$*. By hypothesis, we can now exhibit* Q_0 *s.t.* $Q\xrightarrow{\mu}_{s}\overset{*}{\rightsquigarrow}_{s}Q_0{\downarrow_s}$ *and* $P_0\,\mathcal{R}\,Q_0$*. Using again 3.18 and 3.10, we get that* $\mathcal{T}(Q)\xrightarrow{\mu}_{s}\overset{*}{\rightsquigarrow}_{\sigma}\mathcal{T}(Q_0){\Downarrow_\sigma}$*, and we have* $P'=\mathcal{T}(P_0)\,\mathcal{T}(\mathcal{R})\,\mathcal{T}(Q_0)$*.*

$\underline{\Leftarrow}$: $\mathcal{R}$ *being a* $\sim_{\sigma\downarrow}$*–bisimulation, we know by Lemma 3.10 that* $\mathcal{R}=\mathcal{T}(\mathcal{R}_0)$ *for some* πs *relation* $\mathcal{R}_0$*. Consider now* $(\mathcal{T}(P),\mathcal{T}(Q))\in\mathcal{R}$*, and a process* P' *s.t.* $\mathcal{T}(P)\xrightarrow{\mu}_{\sigma}\overset{*}{\rightsquigarrow}_{\sigma}P'{\downarrow_\sigma}$*; we have again* $P'{\downarrow_\sigma}=\mathcal{T}(P_0{\downarrow_s})$ *for some* P_0*. As above, this implies* $P\xrightarrow{\mu}_{s}\overset{*}{\rightsquigarrow}_{s}P_0{\downarrow_s}$*, and thus by hypothesis there exists* Q_0 *s.t.* $Q\xrightarrow{\mu}_{s}\overset{*}{\rightsquigarrow}_{s}Q_0{\downarrow_s}$ *and* $P_0{\downarrow_s}\,\mathcal{R}_0\,Q_0{\downarrow_s}$*, and* $\mathcal{T}(Q)\xrightarrow{\mu}_{\sigma}\overset{*}{\rightsquigarrow}_{\sigma}\mathcal{T}(Q_0){\downarrow_\sigma}$*, which concludes the proof because* $P'{\downarrow_\sigma}=\mathcal{T}(P_0{\downarrow_s})\,\mathcal{R}\,\mathcal{T}(Q_0{\downarrow_s})$*.* ♢

The results we have established (Theorems 3.14, 3.17 and 3.19) provide great freedom in the strategy for computing manipulations on names, which can lead to the design of several equivalent notions of bisimulation. In particular, one could be interested in avoiding useless computations, that typically arise in presence of *dead code* (e.g. terms of the shape $(\nu a)\,\overline{a}x.P$) or in conjunction with the *bisimulation up to parallel composition proof technique* [San95], where it should be possible to discard some terms without computing the substitutions inside them. Indeed, the call–by–need flavour imposed by the definition of the transition relations suggests a "lazy" computation of relations $\rightsquigarrow$, that could perhaps be performed along the computation of $\xrightarrow{\mu}$ steps in an implementation. Along these lines, the relationship with weak λ–calculi of explicit substitutions [CHL96] could be investigated.

4 Discussion – Variants

We briefly discuss some topics related to our calculi, belonging to variants or extensions of the present work.

4.1 Late Operational Semantics

Not much work has to be done to adapt our study to *late* semantics. We can indeed modify the rules for relations $\xrightarrow{\mu}_{s}$ and $\xrightarrow{\mu}_{\sigma}$ so that the substitution is generated only at synchronisation time. This actually leads to a simpler

presentation[4], because no substitution takes place in the INP rule, so that there is no need to distinguish between bound and free inputs (note that name b in an input action of the shape $a(b)$ is then considered to be bound). We only show here those rules that have to be reformulated in this setting, for πs–calculus:

$$(\text{INP}) \quad a.\lambda P \xrightarrow{a(b)} P$$

$$(\text{COMM}) \quad \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P|Q \xrightarrow{\tau} P'\langle b\rangle^0 \mid Q'} \qquad (\text{CLOSE}) \quad \frac{P \xrightarrow{a(0)} P' \quad Q \xrightarrow{\overline{a}(\nu)} Q'}{P|Q \xrightarrow{\tau} \nu(P'|Q')}$$

4.2 The Polyadic Case

Going from monadic π–calculus to polyadic [Mil91] is not as easy[5]. We present here a preliminary attempt at extending the definition of πs to polyadicity, and leave the complete development of polyadic πs for future work.

Syntax In the polyadic π–calculus, a single communication can involve several names: this means that operator λ becomes polyadic, while ν remains monadic (each application of a restriction being dealt with using rule RES or OPEN). Processes of the polyadic πs–calculus are described by the following syntax:

$$\begin{array}{rclll} a & = & \mathbb{N} & \text{name values} & na = a \mid \mathbf{op}_s\, na & \text{names} \\ \vec{b} & = & [\,] \mid a.\vec{b} & \text{name lists} & \mathbf{op}_s = \langle\vec{b}\rangle^i_k \mid \varphi^i_k \mid \psi_{n,k} & \text{operators} \\ & & P = \mathbf{0} \mid \overline{a}[\vec{b}] \mid a\lambda^k.P \mid (P_1|P_2) \mid \nu P \mid \mathbf{op}_s\, P & & & \text{processes} \end{array}$$

A substitution $\langle\vec{b}\rangle^i_k$ shall sometimes be written $\langle b_1,\ldots,b_n\rangle^i_k$, and ranged over with σ^i_k.

Semantics The syntax of actions has to be modified so that we only have τ and bound receptions and emissions, of the form $\nu^k\, a(\vec{b})$ and $\nu^k\, \overline{a}[\vec{b}]$, the case of free actions being recovered for $k = 0$ (we range over name lists with $\vec{b}$).

[4]However, no conceptual difference arises between both approaches; we preferred an *early* presentation to stay as close as possible to the formalisation of [Hir97] – see below.

[5]In particular, the situation does not resemble the introduction of pairs in the λ–calculus, where a particular shape of terms is added and can take part in a substitution. Here, as terms cannot be exchanged in communication, we have to redesign the mechanism of computation at a deeper level to work with tuples of names instead of names.

Accordingly, the operators introduced by the transition relation are a little more intricate; in particular, as we apply rule RES, the restriction we examine crosses the several (in general) restrictions involved in the action, which means that operator ψ of the πs–calculus now has to perform a circular permutation on indices 0 to k, where k is the number of bound names in the action.

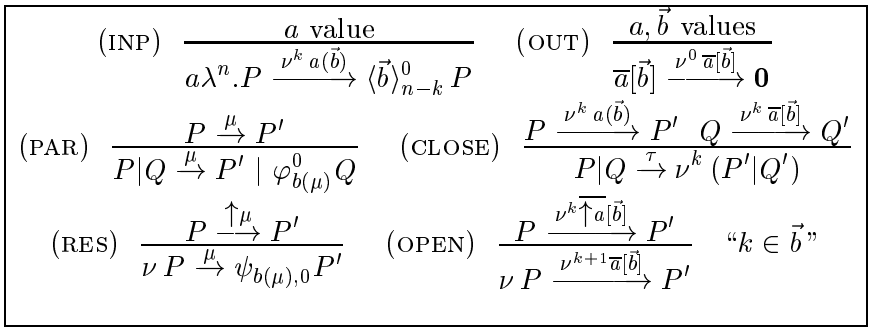


Figure 7: Operational Semantics of Polyadic $\pi\sigma$–calculus

Figure 7 presents the operational semantics of the calculus (relation $\xrightarrow{\mu}_s$; note that since we only work with bound inputs and outputs, we get rid of rules INP_b – which is subsumed by rule INP – and COMM). Note that the notation $\uparrow\mu$ denotes here a more complicate predicate on actions than in the monadic case, as actions may carry several bound names; this holds also for the side condition "$k \in \vec{b}$" for rule OPEN, the details of which we will not enter here. The calculus of operators corresponding to relation $\rightsquigarrow_s$ is defined on Figure 8.

Bisimulation The real difficulty in working in a polyadic setting comes at the level of the definition of behavioural equivalence. Indeed, due to the more complex structure of actions, we face the question of matching two equivalent bound output actions. In the literature, a commonly used abuse of notation allows one to silently permute the names of $\vec{b'}$ in a bound output action of the form $(\nu\vec{b'})\,\overline{a}[\vec{b}]$ (where $\vec{b'} \subseteq \vec{b}$), so that the collection of names $\vec{b'}$ actually has a set structure rather than a vector structure, as the notation suggests. In a De Bruijn setting, however, such an operation is not innocuous, as it changes the representation of the term performing the output action.

σ–destruction	$\langle b_1,\ldots,b_n\rangle^i_k\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad n<i \\ b_{a-i+1}+i & if \quad i\le a<i+n \\ a-k & if \quad i+n\le a \end{cases}$
σ–inp transition	$\sigma^i_k\, a\lambda^n.P$	$\rightsquigarrow_s$	$(\sigma^i_k\, a)\lambda^n.(\sigma^{i+n}_k\, P)$
σ–res transition	$\sigma^i_k\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\sigma^{i+1}_k P)$
σ–out transition	$\sigma^i_k\,(\overline{a}[\vec{b}])$	$\rightsquigarrow_s$	$\overline{\sigma^i_k a}[\sigma^i_k \vec{b}]$
σ–— transition	$\sigma^i_k(P_1\vert P_2)$	$\rightsquigarrow_s$	$\sigma^i_k P_1 \mid \sigma^i_k P_2$
σ–**0** transition	$\sigma^i_k \mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$
φ–destruction	$\varphi^i_k\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad a<i \\ a+k & if \quad a\ge i \end{cases}$
φ–inp transition	$\varphi^i_k\, a\lambda^n.P$	$\rightsquigarrow_s$	$(\varphi^i_k\, a)\lambda^n.(\varphi^{i+n}_k\, P)$
φ–res transition	$\varphi^i_k\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\varphi^{i+1}_k\, P)$
φ–out transition	$\varphi^i_k\,(\overline{a}[\vec{b}])$	$\rightsquigarrow_s$	$\overline{\varphi^i_k a}[\varphi^i_k \vec{b}]$
φ–— transition	$\varphi^i_k\,(P_1\vert P_2)$	$\rightsquigarrow_s$	$\varphi^i_k\, P_1 \mid \varphi^i_k\, P_2$
φ–**0** transition	$\varphi^i_k\, \mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$
ψ–destruction	$\psi_{n,k}\, a$	$\rightsquigarrow_s$	$\begin{cases} a & if \quad a<k \\ a+1 & if \quad k\le a<k+n \\ k & if \quad i=k+n \\ a & if \quad i>k+n \end{cases}$
ψ–inp transition	$\psi_{n,k}\, a\lambda^m.P$	$\rightsquigarrow_s$	$(\psi_{n,k} a)\lambda^m.(\psi_{n+m,k+m} P)$
ψ–res transition	$\psi_{n,k}\, \nu P$	$\rightsquigarrow_s$	$\nu\,(\psi_{n+1,k+1} P)$
ψ–out transition	$\psi_{n,k}\,(\overline{a}[\vec{b}])$	$\rightsquigarrow_s$	$\overline{\psi_{n,k} a}[\psi_{n,k} \vec{b}]$
ψ–— transition	$\psi_{n,k}\,(P_1\vert P_2)$	$\rightsquigarrow_s$	$\psi_{n,k} P_1 \mid \psi_{n,k} P_2$
ψ–**0** transition	$\psi_{n,k}\mathbf{0}$	$\rightsquigarrow_s$	$\mathbf{0}$

Figure 8: Polyadic πs–calculus operators: relation $\rightsquigarrow_s$

As we introduce bisimulation, we want to compare the actions performed by two processes; there are a priori two solutions to tackle the problem of comparing bound output actions (notice that comparing bound input actions is harmless, since the disposition of bound names in an input is not imposed by the shape of the term performing the transition):

- One can introduce an equivalence relation on actions so that the set structure of $\vec{b'}$ is implemented (which amounts to embed in the system the property $(\nu x)(\nu y)\, P \sim (\nu y)(\nu x)\, P$). The transition relation is then defined using this equivalence, which can result in the application of

some operators to the processes coming from a transition. We therefore define a judgment of the form $\mathbf{op}_s \vdash \mu = \mu'$ (actually $\mathbf{op}_s$ will range over an extended syntax for operators w.r.t. the one given above – the extension is however straightforward):

Definition 4.1 (Equality on actions) *The judgment corresponding to equality on actions is defined by the following rules:*

$$id \vdash \mu = \mu \qquad \frac{op \vdash \nu^k \overline{a}[\vec{b}] = \nu^k \overline{a'}[\vec{b'}]}{\psi_{i,j} \circ op \vdash \nu^k \overline{a}[\vec{b}] = \nu^k \overline{(\psi_{i,j} a')}[\psi_{i,j} \vec{b'}]} \quad i,j < k$$

Definition 4.2 (Extended Transition Relation) *We define transition relation $\xrightarrow{\mu}_{s'}$ by adding to the rules of Figure 7 the following transition rule:*

$$\frac{P \xrightarrow{\mu}_s P' \quad op \vdash \mu = \mu'}{P \xrightarrow{\mu'}_{s'} opP'}.$$

Bisimulation is the defined using relations $\xrightarrow{\mu}_{s'}$ and $\rightsquigarrow_s$, as in the monadic case.

- Alternatively, one can define a notion of *canonical form* for bound output actions, so that a *unique* permutation on bound names is chosen for every action, and comparison of actions reduces to syntactic equality. This solution, which we find closer to the De Bruijn approach, involves some quite tedious computation on De Bruijn indices each time rule OPEN is applied, to preserve the property of canonicity for the inferred actions. It is introduced in [Hir99]. For lack of space, we do not enter the details of the corresponding definitions.

4.3 Bisimilarity Proofs and Structural Congruence

While in $\lambda\sigma$ and its variants, confluence properties are a central issue, there is no real notion of such a "benchmark" in our setting. This comes from the fact that the "transition" part of the semantics (relation $\xrightarrow{\mu}$) and the "substitutions" part (relation $\rightsquigarrow$) do not really interfer, which results in the close similarity between $\sim_s$ and $\sim_\sigma$. Such an interplay would however arise if we were to perform *bisimilarity proofs*, which is beyond the scope of this work. In [Hir97], some bisimilarity results, including the laws of *structural congruence*, are mechanically checked using a theorem prover, in a context

that is very close to (polyadic) πs–calculus. This development involves the proof of some technical lemmas relating relations $\xrightarrow{\mu}$ and $\rightsquigarrow$; to quote an example, we establish a property that would be stated as follows in πs:

$$\forall P, P', \mu, i.\ (\varphi^i\, P \xrightarrow{\uparrow\mu}_s \varphi^{i+\mathrm{bn}(\mu)}_{\mathrm{bn}(\mu)} P') \ \Rightarrow\ (P \xrightarrow{\mu}_s P')\,.$$

Considering the large size of the implementation presented in [Hir97] (about 800 lemmas, 75% of them being purely technical), it seems impossible to give a complete account of these proofs on paper, in the setting of the πs– or $\pi\sigma$–calculi.

Let us notice as well that the aforementioned laws of structural congruence, seen as a consequence of the definition of the operational semantics, could serve to make precise the relation with a *reductional semantics* for the π–calculus (that typically leads to a multi–set semantics for processes, akin to the chemical metaphor [BB92]).

5 Conclusion

We have presented two calculi that provide a description of the mechanism of name manipulation in the π–calculus, and shown that they basically describe the same behaviour, leaving great freedom to the user to design a strategy for computation on names.

The $\pi\sigma$–calculus nearly contains the machinery needed in $\lambda\sigma$ (except that we only substitute names for names). It could be interesting to make this observation more precise, possibly by adapting our work to the Blue Calculus [Bou97], that is a supercalculus of both the λ– and the π–calculus. Along the same lines, our treatment of the restriction operator could be adapted to other calculi containing a similar construct, e.g. [Ode94].

As said in Section 1, the definitions of $\sim_s$ and $\sim_\sigma$ are reminiscent of weak bisimilarity, where τ moves are treated as unobservable in the definition of the equivalence. It thus seems natural to introduce $\approx_s$ (and $\approx_\sigma$), by allowing both $\rightsquigarrow_s$ and $\xrightarrow{\tau}_s$ transitions to be fired silently. The study of the equivalence induced by such a definition seems far from being trivial, basically because we would lose the "determinism" (strong normalisation and confluence) of the unobservable computations.

Acknowledgments We would like to thank René Lalement for useful discussions about this work.

References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Amb91] Simon J. Ambler. A de Bruijn notation for the π-calculus. Technical Report 569, Dept. of Computer Science, Queen Mary and Westfield College, London, May 1991.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *TCS*, 96:217–248, 1992.

[Bou97] G. Boudol. The pi-calculus in direct style. In *Proceedings of POPL '97*, pages 228–241, 1997.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[dB72] N.G. de Bruijn. Lambda Calculus Notation with Nameless Dummies: a Tool for Automatic Formula Manipulation, with Application to the Curch-Rosser Theorem. In *Indagationes Mathematicae*, volume 34, pages 381–392. 1972.

[FMQ96] G. Ferrari, U. Montanari, and P. Quaglia. A π–calculus with Explicit Substitutions. *TCS*, 168(1):53–103, November 1996.

[Hir97] D. Hirschkoff. A full formalisation of π-calculus theory in the Calculus of Constructions. In *Proceedings of TPHOL'97*, volume 1275, pages 153–169. LNCS, Springer Verlag, 1997.

[Hir99] D. Hirschkoff. *Mise en œuvre de preuves de bisimulation*. PhD thesis, ENPC, Champs sur Marne, France, January 1999. in french.

[KL98] D. Kesner and P.E. Martínez López. Explicit Substitutions for Objects and Functions. In *Proceedings of PLILP/ALP '98*, number 1490 in LNCS, pages 195–212. Springer Verlag, 1998.

[KR95] F. Kamareddine and A. Ríos. A λ–calculus à la De Bruijn with explicit substitutions. In *Proceedings of PLILP '95*, number 982 in LNCS, pages 45–62. Springer Verlag, 1995.

[LLL98] F. Lang, P. Lescanne, and L. Liquori. A Framework for Defining Object–Calculi. Technical Report 1998–51, LIP, ENS Lyon, 1998.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, LFCS, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[Mil92] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Ode94] M. Odersky. A Functional Theory of Local Names. In *Proceedings of POPL '94*, 1994.

[San95] D. Sangiorgi. On the bisimulation proof method. Revised version of Technical Report ECS–LFCS–94–299, University of Edinburgh, 1994. An extended abstract can be found in Proc. of MFCS'95, LNCS 969, 1995.

[Tur95] D. N. Turner. *The Polymorphic Pi–calculus: Theory and Implementation*. PhD thesis, University of Edimburgh, 1995.