A criterion for separating process calculi

Rosario Pugliese Francesco Tiezzi
Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy
rosario.pugliese@unifi.it tiezzi@dsi.unifi.it

We introduce a new criterion, replacement freeness, to discern the relative expressiveness of process calculi. Intuitively, a calculus is strongly replacement free if replacing, within an enclosing context, a process that cannot perform any visible action by an arbitrary process never inhibits the capability of the resulting process to perform a visible action. We prove that there exists no compositional and interaction sensitive encoding of a not strongly replacement free calculus into any strongly replacement free one. We then define a weaker version of replacement freeness, by only considering replacement of closed processes, and prove that, if we additionally require the encoding to preserve name independence, it is not even possible to encode a non replacement free calculus into a weakly replacement free one. As a consequence of our encodability results, we get that many of the existing process calculi equipped with priority are not replacement free and hence are not encodable into mainstream calculi like CCS and π-calculus, that instead are strongly replacement free. We also prove that variants of π-calculus with match among names, pattern matching or polyadic synchronization are only weakly replacement free, hence they are separated both from process calculi with priority and from mainstream calculi.

Contents

WORKING DRAFT December 29, 2010

1 Introduction

The field of process calculi has been sometimes compared to a 'jungle of interrelated but separate theories' [28], made of plenty of calculi, each one with its own set of concepts, operators, semantics and results. With the aim of turning this jungle into a 'nicely organized garden', many authors have tackled the challenge of devising suitable criteria to classify the different calculi. Relative expressiveness has been then advocated as a valid perspective from which two calculi can be compared. A standard approach is to define a 'proper' encoding of a calculus into another one, i.e. a function mapping terms of the source calculus into terms of the target one that is required to preserve and/or reflect 'reasonably' much of the semantics of the source language and to be structurally defined over its operators. The target calculus is then considered at least as expressive as the source one. Alternatively, one can prove a sort of *separation* result stating that no such encoding exists, thus telling the two calculi apart.

This is an effective approach, but there is no common agreement on which class of encodings has to be used. Several different classes have been introduced (see, e.g., [13, 10, 30, 19, 18, 31, 32, 3, 39, 16]), each one being characterised by the syntactic and semantic properties that the encodings are required to satisfy. Appropriateness of a class depends however from the kind of results one is seeking. Encodings are better, in the sense that they attest that the target calculus has expressive power tighter to that of the source calculus, when satisfying as many properties as possible. Conversely, separation results are stronger and more informative when relying on encodings with minimal requirements.

In this paper we introduce a few criteria and classes of encodings for separating process calculi, and illustrate some results of their application. In particular, we separate extensions of π-calculus from the core calculus and calculi with priority mechanisms from the others.

Since our focus is on separating process calculi, to get more general results, we rely on a minimal set of requirements taken from the literature. The starting point of our investigation are the *reasonable encodings*, introduced in [18] for comparing several communication primitives in the context of π-calculus, We further generalise this already broad class of encodings by dropping the requirements about name invariance, operational correspondence, and divergence preservation and reflection. Thus, we get the class of *basic encodings*, i.e. encodings that are only required to be compositional (the encoding of a compound term is defined by combining the encodings of its sub-terms) and interaction sensitive (the capability to interact with the context through visible actions is preserved and reflected).

We first introduce a new criterion for separating process calculi, named *replacement freeness*. Intuitively, a calculus is strongly replacement free (*strongly rep-free*, for short) if replacing, within an enclosing context, an 'invisible' process (i.e. a process that cannot perform any visible action) by an arbitrary process never inhibits the capability of the resulting process to perform a visible action. We then prove that there exist no basic encodings of non strongly rep-free calculi into strongly rep-free ones.

Intuitively, invisible processes cannot explicitly interact with the enclosing context since they do not perform visible actions (at most, they can only perform 'internal' computation steps). Nevertheless, their behaviour could be implicitly affected by the enclosing context through the generation of substitutions involving the free names of the invisible processes. To prevent also this kind of influence, we will consider the subclass of invisible processes that are closed (i.e. contain no free name) and use it to weaken the condition of replacement freeness. A calculus is hence deemed replacement free (*rep-free*, for short) if replacing, within an enclosing context, a closed invisible process by an arbitrary process never inhibits the capability of the resulting process to perform visible actions. Of course any strongly rep-free calculus is also rep-free; we will show that the converse does not hold. We will call *weakly rep-free* those rep-free calculi that are not strongly rep-free. Since the processes which we now focus on share no free names, we limit ourselves to only consider the subclass of basic encodings that preserve *name*

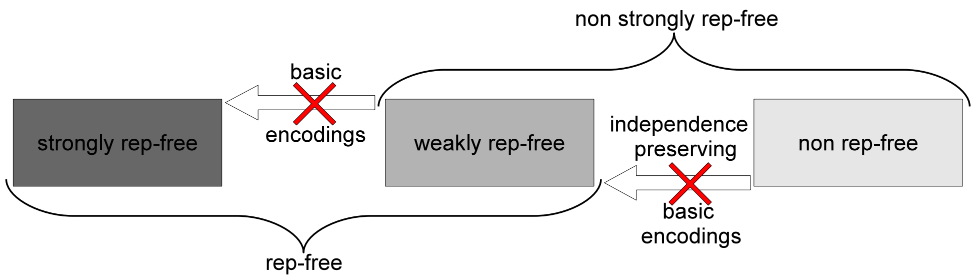

Figure 1: Calculi partition

independence [30, 33] (i.e. if two processes share no free names the same holds for their encodings). We will prove that there exist no such encodings of non rep-free calculi into (even weakly) rep-free calculi. In the end, we obtain a de facto tripartition of process calculi into three sets, i.e. strongly rep-free, weakly rep-free, and non rep-free calculi, which are respectively separated by basic encodings and independence preserving basic encodings (as shown in Figure 1).

Then, we will present several results, arising from the exploitation of our criteria, about the relative expressiveness of well-known process calculi. We first prove that some mainstream process calculi, like CCS [24] and π-calculus [25], are strongly rep-free. Conversely, most of the calculi with some form of priority that have been proposed in the literature (e.g. those in [1, 32, 13, 23], are not rep-free and thus cannot be ‘properly’ encoded into CCS or π-calculus. Intuitively, when replacing an invisible process with one performing prioritized actions, replacement freeness can be violated because the additional initial actions at disposal of the replacing process may prevent some actions with lower priority that might originally be performed. However, this turns out not to be the only source of possible violations of replacement freeness. Indeed, we also show that variants of π-calculus equipped with, respectively, match among names [25], polyadic synchronization [10] or pattern matching [18] are only weakly rep-free and thus are strictly more expressive than the ‘classical’ π-calculus. This allows us to prove in a quite simple and uniform way possibly stronger versions of the results obtained in [32, 10, 19, 18, 39]. As concerns these calculi, strong replacement freeness is violated because a process originally invisible can be transformed into a visible one via application of a name substitution (generated by the enclosing context) which originates a new computation. The reason for this class of violations is clearly different from the previously discussed one and, in fact, if only closed invisible processes are taken into account, these violations do not arise. Thus, the above mentioned richer variants of π-calculus are only weakly rep-free. Anyway, they cannot encode non rep-free calculi like those with priority herein analyzed.

The rest of the paper is structured as follows. Section 2 briefly presents some background notions and the mainstream process calculi CCS and π-calculus. Section 3 introduces the replacement freeness criterion, both in its stronger and in its weaker formulation, and the classes of basic and independence preserving encodings we exploit; it also presents our general separation results. Section 4 proves that CCS and π-calculus are strongly rep-free, Section 5 proves that some variants of π-calculus are only weakly rep-free, and Section 6 proves that several calculi with priority are not (even weakly) rep-free. Finally, Section 7 draws a few conclusions and reviews some strictly related work.

2 Basic notions and process calculi

In this section we introduce the basics concepts we rely on, without committing to any specific formalism. Afterwards, we briefly report syntax and operational semantics of CCS and π-calculus, two well-known process calculi that are our reference frameworks.

Process calculi are formal languages allowing to constructs operational models of open computing systems and to specify interactions between systems. They provide different sets of operators for composing terms, called *processes*, as well as different sets of *(atomic) actions*, typically representing inputs and outputs along communication channels, that processes can perform. Actions are expressed in terms of *names*. We hence assume a countable set of names $\mathscr{N}$, ranged over by letters a, b, $\ldots$, x, y, $\ldots$, n, m, $\ldots$. Names are basic entities without structure. Notationally, when a name is used as a channel, we shall prefer letters $a, b, \ldots$; when a name is used as an input parameter, we shall prefer letters x, y, $\ldots$; to denote a generic name, we shall use letters n, m, $\ldots$. We let $\overline{\mathscr{N}} = \{\overline{n} \mid n \in \mathscr{N}\}$ be the set of *co-names*. $\mathscr{N}$ and $\overline{\mathscr{N}}$ are disjoint and are in bijection via the *complementation* function $\overline{\cdot}$; we define: $\overline{(\overline{n})} = n$.

We also assume that the operational semantics of process calculi is defined by means of *labelled transition systems* (LTSs), that are triples $\langle \mathscr{P}, \mathscr{L}, \longrightarrow \rangle$ where

- $\mathscr{P}$, ranged over by $P, Q, R, P', P_1, \ldots$, is a set of *processes*;
- $\mathscr{L}$, ranged over by $\mu, \mu', \mu_1, \ldots$, is a set of *labels* (of transitions that processes can perform); labels may be either *visible* (we use $\alpha, \alpha', \beta, \ldots$ to range over them) or *invisible* (in which case they are indistinguishable and usually denoted only by τ);
- $\longrightarrow \subseteq \mathscr{P} \times \mathscr{L} \times \mathscr{P}$ is a *transition relation*; we will write $P \xrightarrow{\mu} P'$ to indicate that $(P, \mu, P') \in \longrightarrow$ and say that 'the process P can do a transition labelled μ and become the process P' in doing so' or, shortly, that P' is a μ*-derivative* of P.

Transitions labelled by invisible labels correspond to so-called *computation steps* and can be thought of as taking place of 'internal' interactions of systems, whereas transitions labelled by visible labels can be thought of as representing only 'potential' computation steps, since in order for them to occur they require a contribution from the environment. This means that LTSs account for both how processes can interact with the environment and the activities that their (sub-)processes can perform. Thus, with respect to *reduction relations*, other mathematical structures commonly used to define the operational semantics of process calculi, that only consider computation steps, LTSs better highlight the ability of processes of interacting with their environment.

Due to their special role, transitions labelled by invisible labels are sometimes abstracted from. Notationally, we let $\Longrightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$, $\xRightarrow{\mu}$ to denote $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ (as usual, we write the juxtaposition of two relations to indicate their composition), and $\xRightarrow{\widehat{\mu}}$ to denote $\Longrightarrow$, if $\mu = \tau$, and $\xRightarrow{\mu}$, otherwise. We will write $P \not\xRightarrow{\mu}$ (resp. $P \not\xRightarrow{\widehat{\mu}}$) when there is no process Q such that $P \not\xRightarrow{\mu} Q$ (resp. $P \not\xRightarrow{\widehat{\mu}} Q$). Moreover, we will write $\xrightarrow{\mu}_k$ to denote the composition of $\xrightarrow{\mu}$ with itself k-times (similarly for the other transition relations). By exploiting these relations, we define the following predicates over processes.

Definition 2.1 (Process predicates) *Let P be a process.*

- $P \downarrow_{\mu}$*, i.e. P* can perform a μ transition*, if $P \xrightarrow{\mu} P'$ for some P';*
- $P \Downarrow_{\alpha}$*, i.e. P* can show the (visible) label α*, if $P \xRightarrow{\alpha} P'$ for some P';*
- $P \Downarrow$*, i.e. P is* visible*, if there exists a visible label α such that $P \Downarrow_{\alpha}$;*
- $P \not\Downarrow$*, i.e. P is* invisible*, if there exists no visible label α such that $P \Downarrow_{\alpha}$.*

An invisible process either is stuck or can only perform computation steps. Clearly, if P is invisible and $P \Longrightarrow P'$, then P' is itself invisible. The notion of being invisible will be a central one in the rest of the paper.

To define and delimit the scope of names, process calculi are equipped with *name-binding* operators. An occurrence of a name in a process is *bound* if it is, or it lies within the scope of, a binding occurrence of the name. An occurrence of a name in a process is *free* if it is not bound. We write $\mathtt{fn}(P)$ ($\mathtt{bn}(P)$) for the set of names that have a free (bound) occurrence in P; $\mathtt{n}(P) = \mathtt{fn}(P) \cup \mathtt{bn}(P)$ is the set of *names of* P. We shall write $\mathtt{fn}(P,Q)$ in place of $\mathtt{fn}(P) \cup \mathtt{fn}(Q)$ (similarly for $\mathtt{bn}(\cdot)$ and $\mathtt{n}(\cdot)$). The free names of a process characterize its capability to interact with the environment through execution of transitions with visible labels: e.g. for a name n, in order for P to send/receive via channel n it must be that $n \in \mathtt{fn}(P)$. A process P is *closed* if $\mathtt{fn}(P) = \emptyset$.

In *name-passing* calculi, in which processes can use communications to exchange names, if the scope of bound names can be dynamically 'extruded', the concepts of free and bound names also apply to labels. Furthermore, to express that a process can use the names it receives, syntactic substitutions of names for names are employed. A *substitution* is a function on names that is the identity except on a finite set. We use σ to range over substitutions, and write $n\sigma$ for σ applied to n. We write $\{^{m_1,\ldots,m_k}/_{n_1,\ldots,n_k}\}$ for the substitution σ such that $n_i\sigma = m_i$ and $n_i \neq m_i$, for each i, and $n\sigma = n$ for $n \notin \{n_1,\ldots,n_k\}$. We write $\mathtt{n}(\sigma)$ for the set $\{m_1,\ldots,m_k,n_1,\ldots,n_k\}$ and refer to it as the set of names touched by σ. The effect of applying a substitution σ to a process P, written $P\sigma$, is to replace each free occurrence of each name n in P by $n\sigma$. Substitutions application is usually done in such a way that unintended capture of names by binders is avoided. This may require α*-conversion*, namely the preventive renaming of a finite number of bound names in P. In calculi that admit α-conversion, α-convertible processes are identified; indeed, all the semantics issues will be defined up to α*-equivalence*, that is the processes' equivalence classes induced by α-conversion, also when this is not explicitly mentioned.

Finally, for any given process calculus, we need a notion of *context* where a term of the calculus can be placed for execution. Although we will usually deal with contexts with a single hole, we need to introduce the more general notion of k-hole context.

Definition 2.2 (**k-hole context**) *A k-*hole context*, with $k \geq 1$, is a term of the calculus where k sub-terms are replaced by the holes $\_{}_1$, ..., $\_{}_k$. If C is a k-hole context then we write $C[P_1,\ldots,P_k]$ for the term obtained by replacing $\_{}_i$ in C by P_i, for $i \in [1..k]$.*

We write $\_$ in place of $\_{}_1$ to denote the hole of 1-hole contexts.

From now onward, a name deemed *fresh* in a statement is assumed to be different from any other name there involved, including those names occurring within processes, substitutions and contexts mentioned in the statement, and adopt the following

Convention 2.1 *When considering a set of processes/contexts/substitutions, we assume that the bound names are chosen to be pairwise distinct, different from the free names and not touched by substitutions.*

2.1 CCS

CCS (Calculus of Communicating Systems [24]) is one of the most popular process calculi. It is is based on the notion of process synchronization through execution of complementary actions along the same communication channel. We use names in $\mathscr{N}$ to model input actions along the communication channel having the same name and co-names to model output actions; τ is a distinct invisible action modelling taking place of an internal synchronisation. Thus, the atomic actions that CCS terms can perform are elements of the set $\mathscr{N} \cup \overline{\mathscr{N}} \cup \{\tau\}$, with $\tau \notin \mathscr{N}$. We assume a countable infinite set of *variables*, ranged over by $X, Y, \ldots$. The syntax of CCS is given in Table 1, where $f : \mathscr{N} \to \mathscr{N}$, called *relabelling function*, is a total function such that $\{a \mid f(a) \neq a\}$ is finite, and L is a finite subset of $\mathscr{N}$. A term may be:

P, Q	$::=$		(processes)
		$\mathbf{0}$	(inaction)
	$\mid$	$\mu.P$	(action prefix)
	$\mid$	$P[f]$	(relabelling)
	$\mid$	$P\backslash L$	(restriction)
	$\mid$	$P+Q$	(choice)
	$\mid$	$P \mid Q$	(parallel composition)
	$\mid$	X	(variable)
	$\mid$	$rec\, X.P$	(recursion)

Table 1: CCS syntax

- a term that cannot do anything,
- a term that cannot proceed until it has performed a given action,
- a term where some actions are renamed,
- a term with some actions that cannot be used to interact with the environment,
- a term that can behave as one of two alternative terms according to the enabled actions,
- the parallel composition of two concurrent terms that can proceed independently and can interact via shared channels,
- a variable,
- a recursive term (to also model infinite behaviours).

The restriction operator acts as a name-binder: $P\backslash L$ binds the names in L (and the corresponding co-names) within P. CCS is also equipped with the variable-binding construct $rec\, X.P$, which binds variable X within the scope P. The notion of free and bound occurrences extends to variables as expected. According to the terminology introduced in [24], the terms generated by the grammar in Table 1 should be called *agents*, while agents without occurrences of free variables are deemed *processes*; in the rest of the paper, we will use the word 'process' with exactly the same meaning. Moreover, we will say that P is an *agent in the variable* X to mean that at most the variable X occurs free in P.

The LTS $\langle \mathscr{P}, \mathscr{L}, \longrightarrow \rangle$ defining the operational semantics of CCS is such that the set of processes $\mathscr{P}$ is the set of CCS terms, the set of labels $\mathscr{L}$ is the set of actions (i.e. $\mathscr{N} \cup \overline{\mathscr{N}} \cup \{\tau\}$) and the transition relation $\longrightarrow$ is the least relation induced by the familiar rules reported in Table 2. To define the transition rules, we extend any relabelling function f from $\mathscr{N}$ to $\mathscr{L}$ by letting $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$ for each $\overline{a} \in \overline{\mathscr{N}}$. Moreover, we let $\overline{L} = \{\overline{a} \mid a \in L\}$ for any L finite subset of $\mathscr{N}$.

Notation $P\{^{rec\, X.Q}/_X\}$ denotes substitution within P of the free occurrences of X with $rec\, X.Q$ applying *variable* α-conversion wherever necessary to avoid unintended captures of variables by binders. By definition, application of *variable* substitution $\{^{rec\, X.Q}/_X\}$ to a process commutes with all CCS operators, thus e.g. $(P\backslash L)\{^{rec\, X.Q}/_X\}$ stands for $(P\{^{rec\, X.Q}/_X\})\backslash L$ and $(P[f])\{^{rec\, X.Q}/_X\}$ stands for $(P\{^{rec\, X.Q}/_X\})[f]$, and no *name* α-conversion is applied for avoiding captures. This means that a free occurrence of a name within $rec\, X.Q$ can get bound in $P\{^{rec\, X.Q}/_X\}$ by a restriction of that name whose scope includes X, that is *dynamic scoping* is used for names. As a consequence, we have that name α-equivalence does not hold in CCS (see e.g. [17]).

In the sequel, we will omit trailing occurrences of $\mathbf{0}$, writing μ for $\mu.\mathbf{0}$, and write $[a'_1/a_1, \ldots, a'_n/a_n]$ for the relabelling function f s.t. $f(a) = a'_i$ if $a = a_i$, $i \in \{1, \ldots, n\}$, and $f(a) = a$ otherwise.

$$\frac{}{\mu.P \xrightarrow{\mu} P}\ \textit{(Act)} \qquad \frac{P \xrightarrow{\mu} P'}{P[f] \xrightarrow{f(\mu)} P'[f]}\ \textit{(Rel)} \qquad \frac{P \xrightarrow{\mu} P'}{P\backslash L \xrightarrow{\mu} P'\backslash L}\ \mu \notin L \cup \bar{L}\ \textit{(Res)}$$

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}\ \textit{(Ch-L)} \qquad \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \textit{(Par-L)}$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textit{(Sync)} \qquad \frac{P\{{}^{rec\,X.P}/_{X}\} \xrightarrow{\mu} P'}{rec\,X.P \xrightarrow{\mu} P'}\ \textit{(Rec)}$$

Table 2: Labelled transition rules of CCS (symmetric of rules *(Ch-L)* and *(Par-L)* omitted)

The following results state a few basic properties of the LTS defining the semantics of CCS that will be exploited afterwards.

Lemma 2.1 *Let P, Q and R be CCS processes. Then* $Q \overset{\mu}{\Longrightarrow} P$ *implies* $Q + R \overset{\mu}{\Longrightarrow} P$ *and* $R + Q \overset{\mu}{\Longrightarrow} P$.

Proof. By applying the operational rule *(Ch-L)* and its symmetric (see Table 2), we have that

$$Q \xrightarrow{\mu} S \text{ implies } Q + R \xrightarrow{\mu} S \text{ and } R + Q \xrightarrow{\mu} S$$

which means that every μ-derivative S of Q is also a μ-derivative of $Q + R$ and $R + Q$. The thesis then trivially follows. □

Lemma 2.2 *Let P, Q and R be CCS processes. Then* $Q \overset{\mu}{\Longrightarrow} P$ *implies* $Q \mid R \overset{\mu}{\Longrightarrow} P \mid R$ *and* $R \mid Q \overset{\mu}{\Longrightarrow} R \mid P$.

Proof. By applying the operational rule *(Par-L)* or its symmetric one (see Table 2), we have that

$$Q \xrightarrow{\mu} S \text{ implies } Q \mid R \xrightarrow{\mu} S \mid R \text{ and } R \mid Q \xrightarrow{\mu} R \mid S.$$

The thesis then trivially follows by iterating this reasoning. □

Lemma 2.3 *Let P and Q be CCS processes. Then* $P \overset{\alpha}{\Longrightarrow} P'$ *and* $Q \overset{\overline{\alpha}}{\Longrightarrow} Q'$ *imply* $P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'$.

Proof. From the hypothesis it follows that $P \xrightarrow{\tau}_{m_1} P_1 \xrightarrow{\alpha} P_2 \xrightarrow{\tau}_{m_2} P'$ and $Q \xrightarrow{\tau}_{m_3} Q_1 \xrightarrow{\overline{\alpha}} Q_2 \xrightarrow{\tau}_{m_4} Q'$, for some processes P_1, P_2, Q_1 and Q_2, and natural numbers $m_1, m_2, m_3, m_4 \geq 0$. Now, by applying m_1 times the operational rule *(Par-L)*, m_3 times the symmetric of rule *(Par-L)*, once rule *(Sync)*, m_2 times rule *(Par-L)*, and m_4 times the symmetric of rule *(Par-L)*, we can derive the computation $P \mid Q \xrightarrow{\tau}_{m_1} P_1 \mid Q \xrightarrow{\tau}_{m_3} P_1 \mid Q_1 \xrightarrow{\tau} P_2 \mid Q_2 \xrightarrow{\tau}_{m_2} P' \mid Q_2 \xrightarrow{\tau}_{m_4} P' \mid Q'$, as to be proved. □

2.2 π-calculus

π-calculus [25, 35] is the best known example of name-passing process calculus for modelling concurrent and mobile systems. The novelty with respect to CCS is that processes running concurrently can exchange channels and can thus dynamically modify their scopes. A key distinguishing feature of π-calculus is that it uses a single datatype, *names*, for both values and communication channels. Many slightly different variants of the calculus have been proposed in the literature; here we basically refer to that in [35] but for the 'match' operator (that will be introduced in Section 5).

P,Q	$::=$		(processes)
		$\mathbf{0}$	(inaction)
	$\mid$	$\mu.P$	(action prefix)
	$\mid$	$P+Q$	(choice)
	$\mid$	$P \mid Q$	(parallel composition)
	$\mid$	$(\nu n)P$	(restriction)
	$\mid$	$!P$	(replication)

Table 3: π-calculus syntax

The atomic actions (that processes can perform) are generated by the grammar:

$$\mu \quad ::= \quad a(x) \quad | \quad \overline{a}n \quad | \quad \tau$$

where $a(x)$ means 'input some name along the channel named a and call it x', $\overline{a}n$ means 'output the name n along the channel named a', and τ means taking place of an internal synchronisation. We call a the *subject* and x (resp. n) the *object*. The syntax of π-calculus is given in Table 3. A term may be:

- a term that cannot do anything,
- a term that cannot proceed until it has performed a given action,
- a term that can behave as one of two alternative terms according to the enabled actions,
- the parallel composition of two concurrent terms that can proceed independently and can interact via shared channels,
- a term that declares a new unique name for its own use,
- a term capable of spawning an unbounded number of concurrent copies of itself.

π-calculus has two name-binding operators: the input prefix $a(x).P$, that binds the name x, and the restriction $(\nu n)P$, that binds the name n; in both cases, the scope is P. Differently from the homonymous operator of CCS, restriction in π-calculus, as well as input prefix, uses *static scoping* for names, thus, when a substitution is applied to a process, name α-conversion is performed wherever necessary to avoid unintended captures of names by binders. As a consequence, π-calculus satisfies α-equivalence. These are relevant semantics differences with CCS that uses instead dynamic scoping and does not satisfies α-equivalence. Moreover, in π-calculus the scope of a name can be dynamically extruded by sending the name along some other channel.

When writing processes as linear expressions, we shall usually omit trailing occurrences of $\mathbf{0}$, use parentheses to resolve ambiguity and adopt the conventions that prefixing, restriction, and replication bind more tightly than composition, and prefixing more tightly than choice. Moreover, we regard a substitution as binding more tightly than any process operator.

Let us now present the LTS $\langle \mathscr{P}, \mathscr{L}, \longrightarrow \rangle$ defining the semantics of π-calculus. The set of processes $\mathscr{P}$ is the set of π-calculus terms. The set of labels $\mathscr{L}$ is generated by

$$\mu \quad ::= \quad an \quad | \quad \overline{a}n \quad | \quad \overline{a}(n) \quad | \quad \tau$$

where labels indicate input via a of the name n, output via a of the name n, output via a of the bound name n, and a computation step, respectively. The notions of subject, object, free and bound names,

$$\frac{}{\bar{a}n.P \xrightarrow{\bar{a}n} P}\ \textit{(Out)} \qquad \frac{}{a(x).P \xrightarrow{an} P\{^n/_x\}}\ \textit{(Inp)} \qquad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}\ \textit{(Ch-L)}$$

$$\frac{}{\tau.P \xrightarrow{\tau} P}\ \textit{(Tau)} \qquad \frac{P \xrightarrow{\mu} P'}{(\nu n)P \xrightarrow{\mu} (\nu n)P'}\ n \notin \mathtt{n}(\mu)\ \textit{(Res)} \qquad \frac{P \xrightarrow{\bar{a}n} P'}{(\nu n)P \xrightarrow{\bar{a}(n)} P'}\ n \neq a\ \textit{(Open)}$$

$$\frac{P \xrightarrow{\bar{a}n} P' \quad Q \xrightarrow{an} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \textit{(Comm-L)} \qquad \frac{P \xrightarrow{\bar{a}(n)} P' \quad Q \xrightarrow{an} Q'}{P \mid Q \xrightarrow{\tau} (\nu n)(P' \mid Q')}\ n \notin \mathtt{fn}(Q)\ \textit{(Close-L)}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathtt{bn}(\mu) \cap \mathtt{fn}(Q) = \emptyset\ \textit{(Par-L)} \qquad \frac{P \xrightarrow{\bar{a}n} P' \quad P \xrightarrow{an} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}\ \textit{(Rep-Comm)}$$

$$\frac{P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P' \mid !P}\ \textit{(Rep-Act)} \qquad \frac{P \xrightarrow{\bar{a}(n)} P' \quad P \xrightarrow{an} P''}{!P \xrightarrow{\tau} (\nu n)(P' \mid P'') \mid !P}\ n \notin \mathtt{fn}(P)\ \textit{(Rep-Close)}$$

Table 4: Labelled transition rules of π-calculus (symmetric of rules *(Ch-L)*, *(Comm-L)*, *(Close-L)* and *(Par-L)* omitted)

and substitution application extend to labels as expected. Finally, the transition relation $\longrightarrow$ is the least relation induced by the rules reported in Table 4.

We end by presenting some basic results about the labelled transition relations on π-calculus processes that are fundamental to the theory we will develop in the following sections. The first result is a weaker form of a Lemma from [35] stating some properties of substitutions and transitions.

Lemma 2.4 *Let P be a π-calculus process and σ be a substitution.*

1. *If $P\sigma \xrightarrow{\beta} P'$, where β is $\bar{a}n$ or $\bar{a}(n)$ or am, with m fresh, then $P \xrightarrow{\alpha} P''$ with $\alpha\sigma = \beta$ and $P''\sigma = P'$.*
2. *If $P\sigma \xrightarrow{\tau} P'$ then*
   - *(a) $P \xrightarrow{\tau} P''$ and $P''\sigma = P'$ or*
   - *(b) $P \xrightarrow{\bar{a}n}\xrightarrow{bn} P''$ where $a\sigma = b\sigma$.*
   - *(c) $P \xrightarrow{\bar{a}(n)}\xrightarrow{bn} P''$ where $a\sigma = b\sigma$.*

Proof. Trivially follows from Lemma 1.4.13 in [35]. □

The following results are the analogous for π-calculus of the corresponding ones for CCS. As for CCS, they could be proved by a routine analysis of the LTS's rules applicable on the basis of the terms' leading operator.

Lemma 2.5 *Let P, Q and R be π-calculus processes. Then $Q \overset{\mu}{\Longrightarrow} P$ implies $Q+R \overset{\mu}{\Longrightarrow} P$ and $R+Q \overset{\mu}{\Longrightarrow} P$.*

Lemma 2.6 *Let P, Q and R be π-calculus processes. Then $Q \overset{\mu}{\Longrightarrow} P$ implies $Q \mid R \overset{\mu}{\Longrightarrow} P \mid R$ and $R \mid Q \overset{\mu}{\Longrightarrow} R \mid P$, provided that $\mathtt{bn}(\mu) \cap \mathtt{fn}(R) = \emptyset$.*

Lemma 2.7 *Let P and Q be π-calculus processes. Then $P \overset{\bar{a}n}{\Longrightarrow} P'$ and $Q \overset{an}{\Longrightarrow} Q'$ imply $P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'$ and $Q \mid P \overset{\tau}{\Longrightarrow} Q' \mid P'$.*

Lemma 2.8 *Let P and Q be π-calculus processes. Then $P \overset{\bar{a}(n)}{\Longrightarrow} P'$ and $Q \overset{an}{\Longrightarrow} Q'$ imply $P \mid Q \overset{\tau}{\Longrightarrow} (\nu n)(P' \mid Q')$ and $Q \mid P \overset{\tau}{\Longrightarrow} (\nu n)(Q' \mid P')$, provided that $n \notin \mathtt{fn}(Q)$.*

3 Replacement freeness: a separation criterion

Conceptually speaking, our approach relies on some properties that are invariant under certain classes of encodings, so that, if an encoding violates one such invariant, it cannot belong to the intended class. Strong replacement freeness, or, to be precise, its violation, is an invariant for basic encodings, as well as replacement freeness is an invariant for independence preserving basic encodings. From the quite simple concepts and results presented in this section it follows a powerful methodology for separating process calculi: non strongly rep-free calculi cannot be encoded through basic encodings into calculi that are strongly rep-free. Furthermore, there are no independence preserving basic encodings of non rep-free calculi into (possibly weakly) rep-free ones. In the following sections, we present several results about the relative expressiveness of well-known process calculi arising from the exploitation of our methodology.

As a matter of notation, we shall use $\mathbb{C}$, $\mathbb{C}_1$, $\mathbb{C}_2$, ... to range over process calculi. When convenient, we shall regard a process calculus simply as a set of processes, contexts and operators, writing e.g. $P \in \mathbb{C}$ to mean that process P is an element of $\mathbb{C}$.

3.1 Basic encodings and strong replacement freeness

Basic encodings take their appellation from the minimal properties they are required to satisfy. The first requirement regards the 'structure' of the source calculus: an encoding must be *compositional*, i.e. every $k-$ary operator op of the source calculus is translated into a k-hole context C of the target calculus and the application of op to k processes is encoded into the application of C to the encodings of such processes. The second requirement regards the 'semantics' of the source calculus: an encoding must be *interaction sensitive*, i.e. it must preserve and reflect the capability of a process to perform, or not, visible actions (possibly after some internal computation steps).

Definition 3.1 (Basic encodings) *An encoding $[\![\cdot]\!]$ of $\mathbb{C}_1$ into $\mathbb{C}_2$ is* basic *if*

- $[\![\cdot]\!]$ *is* compositional*: for every $k-$ary operator op in $\mathbb{C}_1$ there is a $k-$hole context $C_{op} \in \mathbb{C}_2$ such that*
$$\forall\, P_1, \ldots,\, P_k \,\in\, \mathbb{C}_1 \qquad [\![op(P_1,\ldots,P_k)]\!] = C_{op}[[\![P_1]\!],\ldots,[\![P_k]\!]]\,.$$
- $[\![\cdot]\!]$ *is* interaction sensitive*: for every process P in $\mathbb{C}_1$, $P \Downarrow$ if and only if $[\![P]\!] \Downarrow$.*

Notice that the property of being an invisible process is an invariant under basic encodings, since by definition such encodings preserve and reflect processes interaction ability.

Proposition 3.1 *Let $[\![\cdot]\!]$ be a basic encoding of $\mathbb{C}_1$ into $\mathbb{C}_2$ and $P \in \mathbb{C}_1$. Then $P \not\Downarrow$ if, and only if, $[\![P]\!] \not\Downarrow$.*

By using a basic encoding (in fact, a compositional one would suffice), one can encode not only single operators but also arbitrary contexts. In other words, any context in $\mathbb{C}_1$ can be represented as a context in $\mathbb{C}_2$. We state, and prove, this property for 1-hole contexts only, but it could be easily generalised.

Lemma 3.1 *Let $[\![\cdot]\!]$ be a basic encoding of $\mathbb{C}_1$ into $\mathbb{C}_2$. Then, for every context $C_1 \in \mathbb{C}_1$, there exists a context $C_2 \in \mathbb{C}_2$ such that, for every process $P \in \mathbb{C}_1$,*

$$[\![C_1[P]]\!] \quad = \quad C_2[[\![P]\!]]. \tag{1}$$

Proof. By induction on the structure of C_1. The base case is when $C_1 = \_$: take $C_2 = \_$ and the thesis is trivially satisfied. In the induction step we have $C_1 = op(P_1,\ldots,P_i,C,P_{i+1},\ldots,P_k)$ for some $k+1$-ary operator op, context C (that, by induction hypothesis, satisfies (1)) and processes $P_1,\ldots,P_k$ (with $k \geq 0$). Then (1) follows since, for every process P, we have:

$$\begin{array}{rcll} [\![C_1[P]]\!] & = & C_{op}[[\![P_1]\!],\ldots,[\![C[P]]\!],\ldots,[\![P_k]\!]] & \text{since } [\![\cdot]\!] \text{ is compositional} \\ & = & C_{op}[Q_1,\ldots,C'[[\![P]\!]],\ldots,Q_k] & \text{by induction and letting} \\ & & & Q_1 = [\![P_1]\!],\ \ldots,\ Q_k = [\![P_k]\!] \\ & = & C_2[[\![P]\!]] & \text{by letting} \\ & & & C_2 = C_{op}[Q_1,\ldots,C',\ldots,Q_k]. \end{array}$$

□

A strongly rep-free calculus is a calculus for which, the replacement within a context of an invisible process with any other one, never inhibits the capability of executing a visible action.

Definition 3.2 (Strong replacement freeness) *A calculus $\mathbb{C}$ is* strongly replacement free *(*strongly rep-free*, for short) if for every context C, invisible process I and process P in $\mathbb{C}$,*

$$C[I] \Downarrow \qquad \textit{implies} \qquad C[P] \Downarrow \tag{2}$$

A calculus is not strongly rep-free if there exists a triple C, I and P violating the condition of Definition 3.2. The proof of the following separation result is based on showing that the property of being a non strongly rep-free calculus is invariant under basic encodings.

Theorem 3.1 *There exists no basic encoding from a non strongly rep-free calculus to a strongly rep-free one.*

Proof. We proceed by contradiction. Let $\mathbb{C}_1$ and $\mathbb{C}_2$ be two process calculi, let $\mathbb{C}_2$ be strongly rep-free and $\mathbb{C}_1$ be not. Let us assume that $[\![\cdot]\!]$ is a basic encoding of $\mathbb{C}_1$ into $\mathbb{C}_2$. Let C, I and P in $\mathbb{C}_1$ be such that $C[I] \Downarrow$ and $C[P] \not\Downarrow$; such a triple exists since $\mathbb{C}_1$ is not strongly rep-free. By Proposition 3.1, $[\![I]\!]$ is invisible. By Lemma 3.1, for some context C' of $\mathbb{C}_2$, we have $[\![C[I]]\!] = C'[[\![I]\!]]$ and $[\![C[P]]\!] = C'[[\![P]\!]]$. Thus, by interaction sensitiveness and Proposition 3.1, $C'[[\![I]\!]] \Downarrow$ and $C'[[\![P]\!]] \not\Downarrow$ against the initial assumption that $\mathbb{C}_2$ is strongly rep-free. □

Remark 3.1 *Our results are equally true for process calculi whose operational semantics is defined by means of reduction relations, instead of LTSs, provided that $\Downarrow_\alpha$ are defined over processes in terms of such manifestations of behaviour as the channels along which a process may communicate with the environment, possibly after a sequence of computation steps. These communication capabilities, called* barbs [26]*, can be defined by induction on the syntax of processes. They provide additional properties that permit to define predicates $\Downarrow$ and $\not\Downarrow$ and uplift the reductional approach.*

3.2 Independence preserving basic encodings and replacement freeness

The set of process calculi violating strong replacement freeness is indeed quite large and can be further split into two distinct sets. One set comprises calculi for which an invisible process I may be transformed into a visible one by application of a substitution σ of names. As shown in Section 5, this happens for calculi exploiting such operators as, e.g., match among names, polyadic synchronization, or pattern matching. The other set includes, at least, those calculi exploiting some form of priority, as shown in Section 6. It is possible to formally separate these two sets of calculi by defining a weaker version of replacement freeness based on the subset of invisible processes that are also closed, that is impervious to substitutions. Intuitively, there is no way for the enclosing context to affect the behaviour of closed invisible processes. We will thus show that the variants of π-calculus presented in Section 5 turn out to be (weakly) rep-free, but not strongly rep-free. Instead, the process calculi with priority presented in Section 6 are not (even weakly) rep-free. To prove separation of these two sets of calculi we only consider those basic encodings that also preserve name independence [30, 33], i.e. guarantee that if two processes do not share free names, the same holds for their encodings (in practice, the encoding does not increase the connectedness of a system of processes).

Definition 3.3 (Name independence)

- *Two processes P and Q are* independent *if* $\mathtt{fn}(P) \cap \mathtt{fn}(Q) = \emptyset$.
- *An encoding $[\![\cdot]\!]$ of $\mathbb{C}_1$ into $\mathbb{C}_2$ is* independence preserving *if whenever processes P and Q in $\mathbb{C}_1$ are independent, then $[\![P]\!]$ and $[\![Q]\!]$ in $\mathbb{C}_2$ are independent too.*

Since closed processes have no free names and substitutions only apply to free names, we get the following obvious, but useful result.

Proposition 3.2 *If P is a closed process, then $P\sigma = P$, for any substitution σ. If I is a closed invisible process, then $I\sigma$ is a closed invisible process too.*

Moreover, we can show that the property of a process to be closed is invariant under independence preserving encodings.

Lemma 3.2 *Let $[\![\cdot]\!]$ be an independence preserving encoding of $\mathbb{C}_1$ into $\mathbb{C}_2$ and let $P \in \mathbb{C}_1$ be a closed process. Then $[\![P]\!] \in \mathbb{C}_2$ is closed too.*

Proof. By contradiction, let P be a closed process such that $[\![P]\!]$ is not closed. Then, by definition of closed process (Definition 3.3), we would get that $\mathtt{fn}(P) = \emptyset$ and, hence, $\mathtt{fn}(P) \cap \mathtt{fn}(P) = \emptyset$. Similarly, we would get that $\mathtt{fn}([\![P]\!]) \neq \emptyset$ and, hence, $\mathtt{fn}([\![P]\!]) \cap \mathtt{fn}([\![P]\!]) \neq \emptyset$. Therefore, we would obtain that processes P and P would be independent while $[\![P]\!]$ and $[\![P]\!]$ would not be, against the hypothesis that the encoding $[\![\cdot]\!]$ is independence preserving. □

Remark 3.2 *We could impose a less demanding condition to basic encodings for separating weakly rep-free calculi from non rep-free ones. In fact, it would suffices to require that closed processes are mapped to closed processes (see Lemma 3.2). We prefer however to resort to a criterion, i.e. name independence, already known in the literature.*

The weak version of replacement freeness is defined by imposing conditions (2) on triples C, I and P, where I is a closed invisible process.

Definition 3.4 (Replacement freeness) *A calculus $\mathbb{C}$ is* replacement free *(*rep-free*, for short) if for every context C, closed invisible process I and process P in $\mathbb{C}$,*

$$C[I] \Downarrow \quad implies \quad C[P] \Downarrow \tag{3}$$

A calculus that is rep-free, but not strongly rep-free is called weakly rep-free.

Of course, every strongly rep-free calculus is also rep-free and every non rep-free calculus is also not strongly rep-free (see Figure 1). Finally, it is possible to obtain the analogous separation result of Theorem 3.1 for rep-free calculi by further requiring the basic encodings to be independence preserving.

Theorem 3.2 *There exists no independence preserving basic encoding from a non rep-free calculus to a rep-free one.*

Proof. The proof proceeds like that of Theorem 3.1 but exploiting the hypothesis that I is closed and that the property of being closed is an invariant under independence preserving encodings (Lemma 3.2). □

Remark 3.3 *When replacing an invisible process I by a generic process P within an enclosing context C, in the case of rep-free calculi, visible actions are preserved and P cannot prevent C from performing actions. Furthermore, in the case of strongly rep-free calculi, C cannot affect I so that it became able to perform visible actions, hence the visible actions of $C[I]$ are necessarily due to C. Instead, in the case of weakly rep-free calculi, by generating name substitutions C can affect I so that it becomes able to perform visible actions, unless I is closed, of course.*

4 Proving strong replacement freeness of mainstream calculi

In this section we prove that CCS and π-calculus are strongly rep-free. We do this by defining a family of relations $\preccurlyeq^k$ for $k < \omega$ (where ω is the first infinite ordinal) and using them for proving, by induction on k, that, both for CCS and π-calculus, it holds that $C[I] \preccurlyeq^\omega C[P]$, for every context C, invisible process I and process P. Proposition 4.5 will then allow us to conclude.

Definition 4.1 (ω-simulation)

1. *$\preccurlyeq^0$ is the universal relation on processes.*
2. *For $0 \le k < \omega$, $\preccurlyeq^k$ is defined by:*
   $Q \preccurlyeq^k P$ if, whenever $Q \xrightarrow{\mu} Q'$, then, for some P', $P \overset{\widehat{\mu}}{\Longrightarrow} P'$ and $Q' \preccurlyeq^{k-1} P'$.
3. *Relation $Q \preccurlyeq^\omega P$ (also referred as ω-*simulation*) holds if $Q \preccurlyeq^k P$ for every k. If $Q \preccurlyeq^\omega P$ then we say that P is an ω-*simulation *of Q or that P ω-*simulates *Q.*

Note that relations $\preccurlyeq^0$, $\preccurlyeq^1$, ..., $\preccurlyeq^\omega$ forms a decreasing sequence. As a matter of notation, we will write $Q \not\preccurlyeq^k P$ (resp. $Q \not\preccurlyeq^\omega P$) when $Q \preccurlyeq^k P$ (resp. $Q \preccurlyeq^\omega P$) does not hold.

Remark 4.1 *The previous relations resemble the 'stratification' of weak bisimulation for π-calculus [35, page 99]. However, our relations are not symmetric and are used as a viable technique for proving that CCS and π-calculus are strongly rep-free, rather than to capture observational equivalences among processes. Given the results in [24, 35] about the stratification of weak bisimulation, we can also presume that, in CCS and π-calculus, $\preccurlyeq^\omega$ coincides with the standard* simulation preorder.

Intuitively, if P ω-simulates Q, then P can perform *at least* the same visible actions that Q can perform, possibly preceded and followed by invisible actions, and the same holds for their derivatives.

Proposition 4.1 *Let P and Q be processes. Then $Q \preccurlyeq^{\omega} P$ implies that for every action μ and process Q' such that $Q \xrightarrow{\mu} Q'$ there exists a process P' such that $P \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^{\omega} P'$.*

Proof. We proceed by contradiction. Suppose that $Q \preccurlyeq^{\omega} P$ and that there exist an action μ and a process Q' such that $Q \xrightarrow{\mu} Q'$ and either $P \overset{\widehat{\mu}}{\not\Longrightarrow}$ or for all processes P' such that $P \xRightarrow{\widehat{\mu}} P'$ we have $Q' \not\preccurlyeq^{\omega} P'$. In the former case, from Definition 4.1, it immediately follows that $Q \not\preccurlyeq^{1} P$, which contradicts the hypothesis $Q \preccurlyeq^{\omega} P$. In the latter case, let $j = max\{k : Q \preccurlyeq^{k} P\}$. Then, there exist j actions $\mu_1, \ldots,$ μ_j and a process Q'' such that $Q' \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_j} Q''$ and for all processes P'' such that $P' \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}}$ P'' it holds that $Q'' \not\preccurlyeq^{1} P''$, i.e. there exists an action μ' and a process Q''' such that $Q'' \xrightarrow{\mu'} Q'''$ and $P'' \overset{\mu'}{\not\Longrightarrow}$. Therefore, we have $Q \xrightarrow{\mu} Q' \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_j} Q'' \xrightarrow{\mu'} Q'''$, and for all processes P'' such that $P \xRightarrow{\widehat{\mu}} P' \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} P''$ it holds that $P'' \overset{\mu'}{\not\Longrightarrow}$. This means that $Q \not\preccurlyeq^{j+1} P$, which again contradicts the hypothesis $Q \preccurlyeq^{\omega} P$. □

By using a similar argument we can prove the following result.

Proposition 4.2 *Let P and Q be processes. Then $Q \preccurlyeq^{k} P$ implies that for every action μ and process Q' such that $Q \xrightarrow{\mu} Q'$ there exists a process P' such that $P \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^{k-1} P'$.*

ω-simulations satisfy some useful properties. They are pre-orders, i.e. they are reflexive and transitive relations, and are in agreement with internal transitions (Lemma 4.2).

Lemma 4.1 *Let P, Q and R be processes, then:*
Reflexivity: *$P \preccurlyeq^{\omega} P$.*
Transitivity: *$Q \preccurlyeq^{\omega} R$ and $R \preccurlyeq^{\omega} P$ imply $Q \preccurlyeq^{\omega} P$.*

Proof. *Reflexivity*. We prove the statement by induction on k. The statement is trivially true for $k = 0$ (by Definition 4.1). Let us assume the statement proved for $\preccurlyeq^{k}$ and prove it for $\preccurlyeq^{k+1}$. If $P \xrightarrow{\mu} P'$, then it immediately follows that $P \xRightarrow{\widehat{\mu}} P'$ and, by induction hypothesis, $P' \preccurlyeq^{k} P'$. This proves that $P \preccurlyeq^{k+1} P$.
Transitivity. We prove that $Q \preccurlyeq^{\omega} R$ and $R \preccurlyeq^{\omega} P$ imply $Q \preccurlyeq^{k} P$ for every k. We proceed by contradiction. Let $j = max\{k : Q \preccurlyeq^{k} P\}$. Then, there exist j actions $\mu_1, \ldots, \mu_j$ and a process Q' such that $Q \xrightarrow{\mu_1}$ $\cdots \xrightarrow{\mu_j} Q'$ and

$$\text{for all processes } P' \text{ such that } P \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} P' \text{ it holds that } Q' \not\preccurlyeq^{1} P', \tag{4}$$

i.e. there exist an action μ and a process Q'' such that $Q' \xrightarrow{\mu} Q''$ and $P' \overset{\mu}{\not\Longrightarrow}$. Since $Q \preccurlyeq^{\omega} R$, by Proposition 4.1, we have that there exists a process R' such that $R \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} R'$ and $Q' \preccurlyeq^{\omega} R'$. Therefore, since $Q' \xrightarrow{\mu} Q''$, again by Proposition 4.1, we have that there exists a process R'' such that $R' \xRightarrow{\widehat{\mu}} R''$ and $Q'' \preccurlyeq^{\omega} R''$. Now, since for all processes P' such that $P \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} P'$ it holds that $P' \overset{\mu}{\not\Longrightarrow}$, we would get that there exists no process P' such that $R' \preccurlyeq^{1} P'$, which contradicts the hypothesis $R \preccurlyeq^{\omega} P$. □

Notice that, although $\preccurlyeq^{\omega}$ is transitive, relations $\preccurlyeq^{k}$ are not, for every $0 \leq k < \omega$ (in fact, the proof of Lemma 4.1(Transitivity) does not rely on the transitivity of relations $\preccurlyeq^{k}$). For example, let $Q = a$, $R = \tau.a$ and $P = \mathbf{0}$; then, we have $Q \preccurlyeq^{1} R$ and $R \preccurlyeq^{1} P$, but, obviously, $Q \preccurlyeq^{1} P$ does not hold. However, a more limited, but still useful, form of transitivity can be proved (by an argument similar to that followed in Lemma 4.1(Transitivity)).

Proposition 4.3 *Let P, Q and R be processes. Then, for every k, $Q \preccurlyeq^{k} R$ and $R \preccurlyeq^{\omega} P$ imply $Q \preccurlyeq^{k} P$.*

Proof. We proceed by contradiction. Suppose that, for some k, $Q \preccurlyeq^{k} R$, $R \preccurlyeq^{\omega} P$ and $Q \preccurlyeq^{k} P$ does not hold. Let $j = max\{i : Q \preccurlyeq^{i} P\}$ (obviously, it should be $j < k$). Then, there exist j actions $\mu_1, \ldots, \mu_j$ and a process Q' such that $Q \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_j} Q'$ and for no process P' such that $P \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} P'$ it holds that $Q' \preccurlyeq^{1} P'$, i.e. there exist an action μ and a process Q'' such that $Q' \xrightarrow{\mu} Q''$ and $P' \not\xRightarrow{\mu}$. Since $Q \preccurlyeq^{k} R$, by Proposition 4.2, we have that there exists a process R' such that $R \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} R'$ and $Q' \preccurlyeq^{k-j} R'$. Therefore, since $Q' \xrightarrow{\mu} Q''$, again by Proposition 4.2, we have that there exists a process R'' such that $R' \xRightarrow{\widehat{\mu}} R''$ and $Q'' \preccurlyeq^{(k-j)-1} R''$. Now, since for all processes P' such that $P \xRightarrow{\widehat{\mu_1}} \cdots \xRightarrow{\widehat{\mu_j}} P'$ it holds that $P' \not\xRightarrow{\mu}$, we would get that there exists no process P' such that $R' \preccurlyeq^{1} P'$, which contradicts the hypothesis $R \preccurlyeq^{\omega} P$. □

Lemma 4.2 *Let P be a process. Then $P \Longrightarrow P'$ implies $P' \preccurlyeq^{\omega} P$.*

Proof. We proceed by induction on k. For $k = 0$ the statement is trivially satisfied by Definition 4.1. Let us assume the statement proved for $\preccurlyeq^{k}$ and prove it for $\preccurlyeq^{k+1}$. If $P = P'$ then the thesis directly follows by reflexivity (Lemma 4.1). Otherwise, suppose that $P' \xrightarrow{\mu} P''$. Then we have $P \Longrightarrow P' \xrightarrow{\mu} P''$. By reflexivity, $P'' \preccurlyeq^{k} P''$ for every k, hence it follows that $P' \preccurlyeq^{k+1} P$ as required. □

Now, we show that any process can ω-simulate any invisible process.

Proposition 4.4 *Let P be a process and I be an invisible process. Then $I \preccurlyeq^{\omega} P$.*

Proof. We proceed by contradiction. If $I \not\preccurlyeq^{\omega} P$ then there exists some k such that $I \not\preccurlyeq^{k} P$. Let $j = max\{k : I \preccurlyeq^{k} P\}$. Then, since I is invisible, there exists a process I' such that $I \xrightarrow{\tau}_j I'$ and for all processes P' such that $P \Longrightarrow P'$ it holds that $I' \not\preccurlyeq^{1} P'$. Therefore, since also I' is invisible, we should have that, for some invisible process I'', $I' \xrightarrow{\tau} I''$ and, for all processes P'' such that $P' \Longrightarrow P''$, $I'' \not\preccurlyeq^{0} P''$, which, of course, contradicts the Definition 4.1. □

Relation $\preccurlyeq^{\omega}$ plays a central role for proving (strongly) replacement freeness, since it provides an inductive proof method. In fact, as the following Proposition states, $C[I] \preccurlyeq^{\omega} C[P]$ readily implies conditions (2) of Definition 3.2 and (3) of Definition 3.4.

Proposition 4.5 (ω-simulations & replacement freeness) *Let $\mathbb{C}$ be a process calculus.*

1. *If, for every context C, invisible process I and process P, it holds that $C[I] \preccurlyeq^{\omega} C[P]$ then $\mathbb{C}$ is a strongly rep-free calculus.*
2. *If, for every context C, closed invisible process I and process P, it holds that $C[I] \preccurlyeq^{\omega} C[P]$ then $\mathbb{C}$ is a rep-free calculus.*

Proof. We only prove the thesis for case *1* as the other case is similar. Let P, I and C be a triple such that $C[I] \preccurlyeq^{\omega} C[P]$ and $C[I] \Downarrow$. To prove that $\mathbb{C}$ is strongly rep-free, we must show that $C[P] \Downarrow$. In fact, $C[I] \Downarrow$ means that $C[I] \xRightarrow{\alpha}$ for some visible action α. Hence, for some $m \geq 0$ and processes Q' and Q'', $C[I] \xrightarrow{\tau}_m Q' \xrightarrow{\alpha} Q''$. Since $C[I] \preccurlyeq^{\omega} C[P]$ implies, in particular, that $C[I] \preccurlyeq^{m+1} C[P]$, then, by applying $m+1$ times Definition 4.1 we get that $C[P] \Longrightarrow_m R' \xRightarrow{\alpha} R''$ for some processes R' and R''. Hence, $C[P] \Downarrow$ and the thesis is proved. $\Box$

In the next subsections we will show that, both for CCS and π-calculus, it holds that $C[I] \preccurlyeq^{\omega} C[P]$, for every context C, invisible process I and process P; thus, both the two calculi are strongly rep-free. Notably, the condition $C[I] \preccurlyeq^{\omega} C[P]$ is stronger than that requested in the (strongly) rep-free definition, since the former requires that if $C[I]$ performs a visible action then $C[P]$ must perform the same action, while the latter simply requires that $C[P]$ is able to perform some visible action. Thus, Proposition 4.5 provides a proof technique, not an alternative characterization of (strongly) replacement freeness.

4.1 CCS is strongly replacement free

We prove that $\preccurlyeq^{\omega}$ is preserved by all the operators of CCS, i.e. the ω-simulation relation is a pre-congruence. This implies that $C[I] \preccurlyeq^{\omega} C[P]$ for every context C, invisible process I and process P, which, by Proposition 4.5(1), suffices to conclude that CCS is a strongly rep-free calculus (Theorem 4.1).

To prove that $\preccurlyeq^{\omega}$ is preserved by all the operators, it is enough to only consider single hole contexts of CCS. Such contexts, according to the syntax of the calculus (reported in Table 1) and to the definition of contexts (Definition 2.2), are generated by the following grammar:

$$C ::= \_ \;\mid\; \mu.C \;\mid\; C\backslash L \;\mid\; C[f] \;\mid\; C+P \;\mid\; P+C \;\mid\; P\,|\,C \;\mid\; C\,|\,P \;\mid\; rec\,X.C$$

We first prove the result for the finite part of the calculus, then we will extend it to the full calculus.

We will exploit the following preliminary lemma.

Lemma 4.3 *Let P, Q and R be CCS processes. Then, for every k, $Q \preccurlyeq^k P$ implies $Q \preccurlyeq^k P+R$ and $Q \preccurlyeq^k R+P$.*

Proof. We proceed by induction on k. For $k=0$ the statement trivially holds by definition of $\preccurlyeq^0$ (Definition 4.1). Now suppose that the statement holds for k, i.e. $Q \preccurlyeq^k P$ implies $Q \preccurlyeq^k P+R$ and $Q \preccurlyeq^k R+P$, and prove it for $k+1$, i.e. $Q \preccurlyeq^{k+1} P$ and $Q \xrightarrow{\mu} Q'$ imply $P+R \xRightarrow{\widehat{\mu}} P'$ and $R+P \xRightarrow{\widehat{\mu}} P'$, for some P', with $Q' \preccurlyeq^k P'$. Since $Q \preccurlyeq^{k+1} P$, we have that, for some P', $P \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^k P'$. Now, if $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by Lemma 2.1, we get $P+R \xRightarrow{\widehat{\mu}} P'$ and $R+P \xRightarrow{\widehat{\mu}} P'$ which proves the thesis. Otherwise, if $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then we have $\mu = \tau$, $P = P'$, and $P+R \xRightarrow{\widehat{\mu}} P+R$ and $R+P \xRightarrow{\widehat{\mu}} R+P$. The thesis follows since, by induction hypothesis, $Q' \preccurlyeq^k P$ implies $Q' \preccurlyeq^k P+R$ and $Q' \preccurlyeq^k R+P$. $\Box$

Proposition 4.6 *Let P, Q and R be CCS processes, μ an action, L a finite subset of input actions and f a relabelling function. Then, for every k, $Q \preccurlyeq^k P$ implies*

1. $\mu.Q \preccurlyeq^k \mu.P$
2. $Q[f] \preccurlyeq^k P[f]$
3. $Q\backslash L \preccurlyeq^k P\backslash L$

4. $Q+R \preccurlyeq^k P+R$
5. $R+Q \preccurlyeq^k R+P$
6. $Q \mid R \preccurlyeq^k P \mid R$
7. $R \mid Q \preccurlyeq^k R \mid P$

Proof. We prove by induction on k that $\preccurlyeq^k$ is preserved for every k. The base case trivially follows by definition of $\preccurlyeq^0$ (Definition 4.1). To prove the induction step, we assume the statement true for k and prove that it also holds for $k+1$. The operational rules we mention in the proof are those defining the semantics of CCS reported in Table 2.

1. $\mu.Q \preccurlyeq^0 \mu.P$ trivially holds by definition of $\preccurlyeq^0$ (Definition 4.1). We prove that, for every k, $Q \preccurlyeq^k P$ implies $\mu.Q \preccurlyeq^{k+1} \mu.P$. Suppose that $\mu.Q \xrightarrow{\mu'} Q'$. Due to the syntactic form of $\mu.Q$, the only applicable operational rule is *(Act)*. This means that $\mu' = \mu$ and $Q' = Q$. By applying the same operational rule, we derive $\mu.P \xrightarrow{\mu} P$ which proves the thesis since, by hypothesis, $Q \preccurlyeq^k P$.

2. Suppose that $Q[f] \xrightarrow{\mu} Q'$. Since the only applicable operational rule is *(Rel)*, this means that, by a shorter proof, $Q \xrightarrow{\mu'} Q''$, with $\mu = f(\mu')$ and $Q' = Q''[f]$. By hypothesis, $Q \preccurlyeq^{k+1} P$, then we get $P \xRightarrow{\widehat{\mu'}} P'$ with $Q'' \preccurlyeq^k P'$. Now, by repeatedly applying the operational rule *(Rel)*, we obtain $P[f] \xRightarrow{\widehat{\mu}} P'[f]$. The thesis then follows since, by induction hypothesis, $Q''[f] \preccurlyeq^k P'[f]$.

3. Suppose that $Q\backslash L \xrightarrow{\mu} Q'$. Since the only applicable operational rule is *(Res)*, this means that, by a shorter proof, $Q \xrightarrow{\mu} Q''$, with $\mu \notin L \cup \overline{L}$ and $Q' = Q''\backslash L$. By hypothesis, $Q \preccurlyeq^{k+1} P$, then we get $P \xRightarrow{\widehat{\mu}} P'$ with $Q'' \preccurlyeq^k P'$. Now, by repeatedly applying the operational rule *(Res)*, we obtain $P\backslash L \xRightarrow{\widehat{\mu}} P'\backslash L$. The thesis then follows since, by induction hypothesis, $Q''\backslash L \preccurlyeq^k P'\backslash L$.

4. Suppose that $Q+R \xrightarrow{\mu} S$. Then, either the operational rule *(Ch-L)* has been applied and, by a shorter proof, $Q \xrightarrow{\mu} S$, or the symmetric rule has been applied and, by a shorter proof, $R \xrightarrow{\mu} S$. In the latter case, by application of the same rule, we get $P+R \xrightarrow{\mu} S$, thus we have $P+R \xRightarrow{\widehat{\mu}} S$ and the thesis follows by reflexivity (Lemma 4.1). In the former case, since $Q \preccurlyeq^{k+1} P$, then $P \xRightarrow{\widehat{\mu}} P'$ and $S \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by Lemma 2.1, we can infer that $P+R \xRightarrow{\widehat{\mu}} P'$ as to be proved (since we already know that $S \preccurlyeq^k P'$). If $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then $\mu = \tau$ and $P = P'$. Thus, we have $P+R \xRightarrow{\widehat{\tau}} P+R$ by the empty sequence of transitions and $S \preccurlyeq^k P+R$ by Lemma 4.3, as to be proved.

5. Similar, but symmetric, to the previous one.

6. Suppose that $Q \mid R \xrightarrow{\mu} S$. Then, due the syntactic form of $Q \mid R$, only one of the following three cases occurs.
   (a) The operational rule *(Par-L)* is applied, thus, by a shorter proof, $Q \xrightarrow{\mu} Q'$ and $S = Q' \mid R$. Since, by hypothesis, $Q \preccurlyeq^{k+1} P$, then $P \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by Lemma 2.2, we can infer that $P \mid R \xRightarrow{\widehat{\mu}} P' \mid R$. Then, the thesis follows since, by induction hypothesis, $Q' \mid R \preccurlyeq^k P' \mid R$. If $\xRightarrow{\widehat{\mu}}$ is the empty sequence

of transitions, then $\mu = \tau$, $P = P'$ and $Q' \preccurlyeq^k P$. Then, the thesis follows since we have $P \mid R \overset{\widehat{\tau}}{\Longrightarrow} P \mid R$, by the empty sequence of transitions, and $Q' \mid R \preccurlyeq^k P \mid R$, by induction hypothesis.

(b) The symmetric of the operational rule *(Par-L)* is applied, thus, by a shorter proof, $R \xrightarrow{\mu} R'$ and $S = Q \mid R'$. By applying again the same operational rule we get $P \mid R \xrightarrow{\mu} P \mid R'$ and the thesis follows as before by induction hypothesis.

(c) The operational rule *(Sync)* is applied, thus $\mu = \tau$ and, for some visible action α, by a shorter proof, we have $Q \xrightarrow{\alpha} Q'$, $R \xrightarrow{\overline{\alpha}} R'$ and $S = Q' \mid R'$. Since $Q \preccurlyeq^{k+1} P$, we have $P \overset{\alpha}{\Longrightarrow} P'$ (as α is visible then $\overset{\widehat{\alpha}}{\Longrightarrow}$ coincides with $\overset{\alpha}{\Longrightarrow}$), and $Q' \preccurlyeq^k P'$. Now, by Lemma 2.3, we get $P \mid R \overset{\tau}{\Longrightarrow} P' \mid R'$. Moreover, by induction hypothesis, we get $Q' \mid R' \preccurlyeq^k P' \mid R'$, and the thesis is proved.

7. Similar, but symmetric, to the previous one.

□

From the previous result it follows that, unlike weak bisimulation [24], the $\omega-$simulation is preserved by the choice operator. Although this also holds for the parallel composition operator, in this case, to deal with possibly recursive processes, we will need the following stronger result.

Lemma 4.4 *Let P_1, P_2, Q_1 and Q_2 be CCS processes. Then, for every k, $Q_1 \preccurlyeq^k Q_2$ and $P_1 \preccurlyeq^k P_2$ imply $Q_1 \mid P_1 \preccurlyeq^k Q_2 \mid P_2$.*

Proof. We proceed by induction on k. The base case trivially follows by definition of $\preccurlyeq^0$ (Definition 4.1). To prove the induction step, we assume the statement true for k and prove that it also holds for $k+1$. The operational rules we mention in the proof are those defining the semantics of CCS reported in Table 2. Thus, suppose that $Q_1 \preccurlyeq^k Q_2$ and $P_1 \preccurlyeq^k P_2$ imply $Q_1 \mid P_1 \preccurlyeq^k Q_2 \mid P_2$, that $Q_1 \preccurlyeq^{k+1} Q_2$ and $P_1 \preccurlyeq^{k+1} P_2$, and that $Q_1 \mid P_1 \xrightarrow{\mu} R$. We must show that, for some S, $Q_2 \mid P_2 \overset{\widehat{\mu}}{\Longrightarrow} S$ and $R \preccurlyeq^k S$. Due the syntactic form of $Q_1 \mid P_1$, only one of the following three cases occurs.

1. The operational rule *(Par-L)* is applied, thus, by a shorter proof, $Q_1 \xrightarrow{\mu} Q_1'$ and $R = Q_1' \mid P_1$. Since, by hypothesis, $Q_1 \preccurlyeq^{k+1} Q_2$, then $Q_2 \overset{\widehat{\mu}}{\Longrightarrow} Q_2'$ and $Q_1' \preccurlyeq^k Q_2'$. If $\overset{\widehat{\mu}}{\Longrightarrow}$ is not the empty sequence of transitions, then, by Lemma 2.2, we can infer that $Q_2 \mid P_2 \overset{\widehat{\mu}}{\Longrightarrow} Q_2' \mid P_2$. Then, the thesis follows since, by induction hypothesis, $Q_1' \mid P_1 \preccurlyeq^k Q_2' \mid P_2$. If $\overset{\widehat{\mu}}{\Longrightarrow}$ is the empty sequence of transitions, then $\mu = \tau$, $Q_2 = Q_2'$ and $Q_1' \preccurlyeq^k Q_2$. Then, the thesis follows since we have $Q_2 \mid P_2 \overset{\widehat{\tau}}{\Longrightarrow} Q_2 \mid P_2$, by the empty sequence of transitions, and $Q_1' \mid P_1 \preccurlyeq^k Q_2 \mid P_2$, by induction hypothesis.

2. The symmetric of the operational rule *(Par-L)* is applied, thus, by a shorter proof, $P_1 \xrightarrow{\mu} P_1'$ and $R = Q_1 \mid P_1'$. This case is similar, but symmetric, to the previous one.

3. The operational rule *(Sync)* is applied, thus $\mu = \tau$ and, for some visible action α, by a shorter proof, we have $Q_1 \xrightarrow{\alpha} Q_1'$, $P_1 \xrightarrow{\overline{\alpha}} P_1'$ and $R = Q_1' \mid P_1'$. Since $Q_1 \preccurlyeq^{k+1} Q_2$, we have $Q_2 \overset{\alpha}{\Longrightarrow} Q_2'$ (as α is visible then $\overset{\widehat{\alpha}}{\Longrightarrow}$ coincides with $\overset{\alpha}{\Longrightarrow}$), and $Q_1' \preccurlyeq^k Q_2'$. Similarly, since $P_1 \preccurlyeq^{k+1} P_2$, we have $P_2 \overset{\overline{\alpha}}{\Longrightarrow} P_2'$, and $P_1' \preccurlyeq^k P_2'$. Now, by Lemma 2.3, we get $Q_2 \mid P_2 \overset{\tau}{\Longrightarrow} Q_2' \mid P_2'$. Moreover, by induction hypothesis, we get that $Q_1' \mid P_1' \preccurlyeq^k Q_2' \mid P_2'$, and the thesis is proved.

□

To prove that $\preccurlyeq^{\omega}$ is also preserved by recursive definitions, we follow the same strategy adopted in [24] for proving that strong bisimulation is preserved by recursive definitions. Therefore, we extend the definition of ω-simulations to CCS agents as follows.

Definition 4.2 *Let P and Q be two agents in the variable X. We say that $Q \preccurlyeq^k P$ (resp. $Q \preccurlyeq^{\omega} P$) if, for every process R, $Q\{^R/_X\} \preccurlyeq^k P\{^R/_X\}$ (resp. $Q\{^R/_X\} \preccurlyeq^{\omega} P\{^R/_X\}$).*

We will use the following auxiliary result.

Lemma 4.5 *For every agent P in the variable X, $P\{^{rec\,X.P}/_X\} \preccurlyeq^{\omega} rec\,X.P$.*

Proof. We have to prove that $P\{^{rec\,X.P}/_X\} \preccurlyeq^k rec\,X.P$ for every k. Relation $P\{^{rec\,X.P}/_X\} \preccurlyeq^0 rec\,X.P$ trivially follows from the definition of $\preccurlyeq^0$ (Definition 4.1). For $k > 0$, the thesis follows since, by the operational rule *(Rec)* in Table 2, $P\{^{rec\,X.P}/_X\} \xrightarrow{\mu} Q$ implies $rec\,X.P \xrightarrow{\mu} Q$, and, by reflexivity (Lemma 4.1), we have that $Q \preccurlyeq^{k-1} Q$. □

Proposition 4.7 *Let P and Q be two agents in the variable X. If $Q \preccurlyeq^{\omega} P$ then $rec\,X.Q \preccurlyeq^{\omega} rec\,X.P$.*

Proof. We will prove that, if G is an agent in the variable X, then $G\{^{rec\,X.Q}/_X\} \preccurlyeq^k G\{^{rec\,X.P}/_X\}$ for every k. From this, by setting $G = X$, we obtain that $rec\,X.Q \preccurlyeq^{\omega} rec\,X.P$ as to be proved. We proceed by induction on k. The base case $G\{^{rec\,X.Q}/_X\} \preccurlyeq^0 G\{^{rec\,X.P}/_X\}$ trivially follows from the definition of $\preccurlyeq^0$ (Definition 4.1). Let then assume the statement proved for k and prove it for $k+1$. Thus, assuming that $G\{^{rec\,X.Q}/_X\} \preccurlyeq^k G\{^{rec\,X.P}/_X\}$, we have to prove that, if $G\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} Q'$, then $G\{^{rec\,X.P}/_X\} \overset{\widehat{\mu}}{\Longrightarrow} P'$ with $Q' \preccurlyeq^k P'$. We prove this statement by transition induction on the depth of the inference by which the transition $G\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} Q'$ is inferred. We proceed by cases on the structure of G.

1. $G = X$. Then $G\{^{rec\,X.Q}/_X\} = rec\,X.Q$, hence $rec\,X.Q \xrightarrow{\mu} Q'$. Since the only applicable operational rule is *(Rec)*, this means that $Q\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} Q'$ by a shorter proof. Hence, by induction on the depth of the inference, $Q\{^{rec\,X.P}/_X\} \overset{\widehat{\mu}}{\Longrightarrow} Q''$ and $Q' \preccurlyeq^k Q''$. This means that either (i) $Q\{^{rec\,X.P}/_X\} \xrightarrow{\tau}_{m_1} Q_1 \xrightarrow{\mu} Q_2 \xrightarrow{\tau}_{m_2} Q''$ for some $m_1, m_2 \geq 0$, Q_1 and Q_2, or (ii) $\overset{\widehat{\mu}}{\Longrightarrow}$ is the empty sequence of transitions, thus $\mu = \tau$ and $Q'' = Q\{^{rec\,X.P}/_X\}$. In case (i), since $Q \preccurlyeq^{\omega} P$, by applying $m_1 + m_2 + 1$ times Definition 4.1, we obtain that $P\{^{rec\,X.P}/_X\} \Longrightarrow_{m_1} P_1 \overset{\mu}{\Longrightarrow} P_2 \Longrightarrow_{m_2} P'$ and, by Proposition 4.1, $Q'' \preccurlyeq^{\omega} P'$. In case (ii), since $Q \preccurlyeq^{\omega} P$, if we let $P' = P\{^{rec\,X.P}/_X\}$ then we get $Q'' = Q\{^{rec\,X.P}/_X\} \preccurlyeq^{\omega} P\{^{rec\,X.P}/_X\} = P'$. In both cases, by Proposition 4.3, we get $Q' \preccurlyeq^k P'$ as required.

2. $G = \mu_1.G_1$. Since $(\mu_1.G_1)\{^{rec\,X.Q}/_X\} = \mu_1.(G_1\{^{rec\,X.Q}/_X\})$, then the only applicable operational rule is *(Act)* thus it must be $\mu_1 = \mu$ and $Q' = G_1\{^{rec\,X.Q}/_X\}$. By the same rule, we have $\mu.(G_1\{^{rec\,X.P}/_X\}) \xrightarrow{\mu} G_1\{^{rec\,X.P}/_X\}$. The thesis then follows by taking $P' = G_1\{^{rec\,X.P}/_X\}$ since, by induction hypothesis on k, $Q' = G_1\{^{rec\,X.Q}/_X\} \preccurlyeq^k G_1\{^{rec\,X.P}/_X\} = P'$.

3. $G = G_1[f]$, where f is a relabelling function. Because of the syntactic form of the term, the only applicable operational rule is *(Rel)* thus it must be $\mu = f(\mu_1)$, $Q' = G'[f]$ and $G_1\{^{rec\,X.Q}/_X\} \xrightarrow{\mu_1} G'$ by a shorter proof. Then, by induction on the depth of the inference, $G_1\{^{rec\,X.P}/_X\} \overset{\widehat{\mu_1}}{\Longrightarrow} G''$ and $G' \preccurlyeq^k G''$. Hence we have $G_1[f]\{^{rec\,X.P}/_X\} \overset{\widehat{\mu}}{\Longrightarrow} G''[f]$, by applying rule *(Rel)*, and $G'[f] \preccurlyeq^k G''[f]$, by Proposition 4.6. The thesis follows by taking $P' = G''[f]$.

4. $G = G_1\backslash L$ with L finite subset of input actions. Because of the syntactic form of the term, the only applicable operational rule is *(Res)* thus $\mu \notin L \cup \overline{L}$, $Q' = G'\backslash L$ and $G_1\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} G'$ by a shorter proof. Then, by induction on the depth of the inference, $G_1\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} G''$ and $G' \preccurlyeq^k G''$. Hence we have $G_1\backslash L\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} G''\backslash L$, by applying rule *(Res)*, and $G''\backslash L \preccurlyeq^k G'\backslash L$, by Proposition 4.6. The thesis then follows by taking $P' = G''\backslash L$.
5. $G = G_1 + G_2$. Because of the syntactic form of the term, the only applicable operational rules are *(Ch-L)* and its symmetric one, thus, by a shorter proof, we have $G_1\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} Q'$ or $G_2\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} Q'$. We only show the former case, since the latter is similar. By induction on the depth of the inference, we get $G_1\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, by Lemma 2.1, we get $(G_1+G_2)\{^{rec\,X.P}/_X\} = G_1\{^{rec\,X.P}/_X\} + G_2\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} P'$ and the thesis follows since we already know that $Q' \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then we have $\mu = \tau$ and $P' = G_1\{^{rec\,X.P}/_X\}$. Then, we have $(G_1+G_2)\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} (G_1+G_2)\{^{rec\,X.P}/_X\}$ and, by Lemma 4.3, $Q' \preccurlyeq^k G_1\{^{rec\,X.P}/_X\} + G_2\{^{rec\,X.P}/_X\} = (G_1+G_2)\{^{rec\,X.P}/_X\}$, which proves the thesis.
6. $G = G_1 \mid G_2$. Because of the syntactic form of the term, the only applicable operational rules are *(Par-L)*, its symmetric form and *(Sync)*, thus we have, respectively
   (a) $Q' = G_1' \mid G_2\{^{rec\,X.Q}/_X\}$ with $G_1\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} G_1'$ by a shorter proof;
   (b) $Q' = G_1\{^{rec\,X.Q}/_X\} \mid G_2'$ with $G_2\{^{rec\,X.Q}/_X\} \xrightarrow{\mu} G_2'$ by a shorter proof;
   (c) $Q' = G_1' \mid G_2'$ with $\mu = \tau$ and, for some visible action α, $G_1\{^{rec\,X.Q}/_X\} \xrightarrow{\alpha} G_1'$ and $G_2\{^{rec\,X.Q}/_X\} \xrightarrow{\overline{\alpha}} G_2'$ by shorter proofs.

   In the first case, by induction on the depth of the inference, we have $G_1\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} G_1''$ and $G_1' \preccurlyeq^k G_1''$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, by Lemma 2.2, we get $(G_1 \mid G_2)\{^{rec\,X.P}/_X\} = G_1\{^{rec\,X.P}/_X\} \mid G_2\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} G_1'' \mid G_2\{^{rec\,X.P}/_X\}$. By induction hypothesis on k, we have $G_2\{^{rec\,X.Q}/_X\} \preccurlyeq^k G_2\{^{rec\,X.P}/_X\}$. Thus, by Lemma 4.4, we get $G_1' \mid G_2\{^{rec\,X.Q}/_X\} \preccurlyeq^k G_1'' \mid G_2\{^{rec\,X.P}/_X\}$, as to be proved. If $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then we have $\mu = \tau$, $G_1'' = G_1\{^{rec\,X.P}/_X\}$ and $(G_1 \mid G_2)\{^{rec\,X.P}/_X\} \xRightarrow{\widehat{\mu}} (G_1 \mid G_2)\{^{rec\,X.P}/_X\}$. Thus, by Lemma 4.4, we get $G_1' \mid G_2\{^{rec\,X.Q}/_X\} \preccurlyeq^k G_1\{^{rec\,X.P}/_X\} \mid G_2\{^{rec\,X.P}/_X\} = (G_1 \mid G_2)\{^{rec\,X.P}/_X\}$, which proves the thesis. The second case is specular to the first one. In the third case, by induction on the depth of the inference, we have $G_1\{^{rec\,X.P}/_X\} \xRightarrow{\alpha} G_1''$, with $G_1' \preccurlyeq^k G_1''$, and $G_2\{^{rec\,X.P}/_X\} \xRightarrow{\overline{\alpha}} G_2''$, with $G_2' \preccurlyeq^k G_2''$. By Lemma 2.3, we get $(G_1 \mid G_2)\{^{rec\,X.P}/_X\} \xRightarrow{\tau} G_1'' \mid G_2''$. Finally, by Lemma 4.4, we get $G_1' \mid G_2' \preccurlyeq^k G_1'' \mid G_2''$ as required.
7. $G = rec\,Y.G_1$. Let $G_Q = G_1\{^{rec\,X.Q}/_X\}$ and $G_P = G_1\{^{rec\,X.P}/_X\}$. Because of the syntactic form of the term, the only applicable operational rule is *(Rec)*, thus $G_Q\{^{rec\,Y.G_Q}/_Y\} \xrightarrow{\mu} Q'$ by a shorter proof. Then, by induction on the depth of the inference, $G_P\{^{rec\,Y.G_P}/_Y\} \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by applying rule *(Rec)*, we obtain $rec\,Y.G_P \xRightarrow{\widehat{\mu}} P'$ as required (since we already know that $Q' \preccurlyeq^k P'$). Otherwise, $\mu = \tau$ and $P' = G_P\{^{rec\,Y.G_P}/_Y\}$. By Lemma 4.5, we have $P' \preccurlyeq^\omega rec\,Y.G_P$. From $Q' \preccurlyeq^k P'$ and $P' \preccurlyeq^\omega rec\,Y.G_P$, by Proposition 4.3, we get $Q' \preccurlyeq^k rec\,Y.G_P$ as to be proved.

□

Lemma 4.1 and Propositions 4.6 and 4.7 allow us to conclude that $\preccurlyeq^{\omega}$ is a pre-congruence.

Corollary 4.1 *Relation $\preccurlyeq^{\omega}$ is a pre-congruence for CCS.*

It is now easy to prove that CCS is strongly rep-free.

Theorem 4.1 *CCS is strongly rep-free.*

Proof. Let C be a context, I be an invisible process, and P be a process. By Proposition 4.4, we have that $I \preccurlyeq^{\omega} P$. By Corollary 4.1, we conclude that $C[I] \preccurlyeq^{\omega} C[P]$. Then, by Proposition 4.5(1), it follows that CCS is a strongly rep-free calculus. □

4.2 π-calculus is strongly replacement free

To prove that π-calculus is a strongly rep-free calculus (Theorem 4.2), we proceed as for CCS. However, the ω-simulation relation results to be a pre-congruence w.r.t all the operators of π-calculus but for the input prefix. Another difference with the previous section is the proof technique used.

According to the syntax of the calculus, that is reported in Table 3, and to the definition of contexts (Definition 2.2), the contexts of π-calculus we have to consider are generated by the following grammar:

$$C ::= \_ \quad | \quad \mu.C \quad | \quad C+P \quad | \quad P+C \quad | \quad P\,|\,C \quad | \quad C\,|\,P \quad | \quad (\nu n)C \quad | \quad !C$$

Unfortunately, $\preccurlyeq^{\omega}$ is not a pre-congruence for π-calculus. This can be easily seen by means of an example. Let $C = a(x).\_$, $Q = x(y).\bar{c}y \mid \bar{b}d$ and $P = x(y).(\bar{c}y \mid \bar{b}d) + \bar{b}d.x(y).\bar{c}y$. Then, we have $Q \preccurlyeq^{\omega} P$, but $C[Q] \not\preccurlyeq^{\omega} C[P]$. In fact, we have $C[Q] \xrightarrow{ab} b(y).\bar{c}y \mid \bar{b}d \xrightarrow{\tau} \bar{c}d \mid \mathbf{0} \xrightarrow{\bar{c}d} \mathbf{0} \mid \mathbf{0}$ while $C[P] \xrightarrow{ab} b(y).(\bar{c}y \mid \bar{b}d) + \bar{b}d.b(y).\bar{c}y = P'$ and $P' \overset{\bar{c}d}{\not\Longrightarrow}$. Intuitively, execution of the input action along channel a generates the substitution $\{b/x\}$ that identifies the names x and b that were originally different. As a consequence of application of the substitution to the continuation process, new transitions can arise because communications that were originally impossible become enabled (which paves the way for further new actions). This is exactly what happens as for process Q, since the application of the substitution makes channels x and b equal and allows synchronization between the two parallel input and output actions along b. The same does not happen for process P, as in this case the two complementary actions are sequentialized.

Indeed, $\preccurlyeq^{\omega}$ is preserved by all operators but input prefix, as we are going to show. To this aim we exploit the fact that our ω-simulation is strictly weaker than a well-known observational semantics for π-calculus, named *weak bisimilarity* [25, 35] and denoted by $\approx$. This fact can be easily proved by relying on a family of relations $\approx_k$, that stratify the definition of weak bisimilarity and whose intersection $\approx_{\omega}$ is weaker than $\approx$ but stronger than $\preccurlyeq^{\omega}$. We report below the definition (from [35, Def. 2.4.24, page 99]) of this family of relations.

Definition 4.3 (Stratification of $\approx$)

1. *$\approx_0$ is the universal relation on processes.*
2. *For $0 \le k < \omega$, $\approx_k$ is defined by: $Q \approx_k P$ if,*
   - *whenever $Q \xrightarrow{\mu} Q'$, then, for some P', $P \overset{\widehat{\mu}}{\Longrightarrow} P'$ and $Q' \approx_{k-1} P'$*
   - *whenever $P \xrightarrow{\mu} P'$, then, for some Q', $Q \overset{\widehat{\mu}}{\Longrightarrow} Q'$ and $Q' \approx_{k-1} P'$.*

3. *Relation $Q \approx_\omega P$ holds if $Q \approx_k P$ for every k.*

Now, from the definition above, it is trivial to see that, for every k, we have $\approx_k \subseteq \preccurlyeq^k$ and that this implies that $\approx_\omega \subseteq \preccurlyeq^\omega$. Therefore, since in [35, Theorem 2.4.27, page 100] it is proven that $\approx$ implies $\approx_\omega$, we have that, for every pair of π-calculus processes P and Q, $Q \approx P$ implies $Q \preccurlyeq^\omega P$ and $P \preccurlyeq^\omega Q$. Thus, all the laws that are sound for $\approx$ (as e.g. those in [25, 35]) do also hold for $\preccurlyeq^\omega$. In particular, as stated by the following lemma, we have that replication 'absorbs' parallel additional copies of the replicated process.

Lemma 4.6 *Let P be a π-calculus process. Then $P \,|\, !P \preccurlyeq^\omega !P$.*

Now, we can prove that all π-calculus operators, but input prefix, preserve $\preccurlyeq^\omega$.

Proposition 4.8 *Let P, Q and R be π-calculus processes, and a and n be names. Then, for every k, $Q \preccurlyeq^k P$ implies*

1. $\tau.Q \preccurlyeq^k \tau.P$
2. $\overline{a}n.Q \preccurlyeq^k \overline{a}n.P$
3. $Q+R \preccurlyeq^k P+R$
4. $R+Q \preccurlyeq^k R+P$
5. $Q \,|\, R \preccurlyeq^k P \,|\, R$
6. $R \,|\, Q \preccurlyeq^k R \,|\, P$
7. $(\nu n)Q \preccurlyeq^k (\nu n)P$
8. $!Q \preccurlyeq^k !P$.

Proof. The statement derives, by reflexivity (Lemma 4.1), from the following stronger result we are going to prove.

Let P_1, P_2, Q_1 and Q_2 be π-calculus processes, and a and n be names. Then, for every k, $Q_1 \preccurlyeq^k P_1$ and $Q_2 \preccurlyeq^k P_2$ imply

1. $\tau.Q_1 \preccurlyeq^k \tau.P_1$;
2. $\overline{a}n.Q_1 \preccurlyeq^k \overline{a}n.P_1$;
3. $Q_1 \preccurlyeq^k P_1 + R$ and $Q_1 \preccurlyeq^k R + P_1$, for some process R;
4. $Q_1 + P_1 \preccurlyeq^k Q_2 + P_2$;
5. $Q_1 \,|\, P_1 \preccurlyeq^k Q_2 \,|\, P_2$;
6. $(\nu n)Q_1 \preccurlyeq^k (\nu n)P_1$;
7. $!Q_1 \preccurlyeq^k !P_1$.

We prove by induction on k that $\preccurlyeq^k$ is preserved for every k. The base case trivially follows by definition of $\preccurlyeq^0$ (Definition 4.1). To prove the induction step, we assume the statement true for k and prove that it also holds for $k+1$. The operational rules we mention in the proof are those defining the semantics of π-calculus reported in Table 4.

1. Suppose that $\tau.Q_1 \xrightarrow{\mu} Q_1'$. Due to the syntactic form of the term, the only applicable operational rule is *(Tau)*, thus we have $\mu = \tau$ and $Q_1' = Q_1$. By applying the same operational rule, we derive $\tau.P_1 \xrightarrow{\tau} P_1$ which proves the thesis since, by hypothesis, $Q_1 \preccurlyeq^{k+1} P_1$.
2. The proof is similar to the previous one, but the operational rule *(Out)* is used in place of *(Tau)*.

3. This is the analogous for π-calculus of Lemma 4.3. Suppose that $Q_1 \xRightarrow{\widehat{\mu}} Q'$. Since $Q_1 \preccurlyeq^{k+1} P_1$, we have that, for some P', $P_1 \xRightarrow{\widehat{\mu}} P'$ and $Q' \preccurlyeq^{k} P'$. Now, if $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by Lemma 2.5, we get $P_1 + R \xRightarrow{\widehat{\mu}} P'$ and $R + P_1 \xRightarrow{\widehat{\mu}} P'$ which proves the thesis. Otherwise, if $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then we have $\mu = \tau$, $P_1 = P'$, and $P_1 + R \xRightarrow{\widehat{\mu}} P_1 + R$ and $R + P_1 \xRightarrow{\widehat{\mu}} R + P_1$. The thesis follows since, by induction hypothesis, $Q' \preccurlyeq^{k} P_1$ implies $Q' \preccurlyeq^{k} P_1 + R$ and $Q' \preccurlyeq^{k} R + P_1$.

4. Suppose that $Q_1 + P_1 \xrightarrow{\mu} S$. Then, either the operational rule *(Ch-L)* has been applied and, by a shorter proof, $Q_1 \xrightarrow{\mu} S$, or the symmetric rule has been applied and, by a shorter proof, $P_1 \xrightarrow{\mu} S$. In the former case, since $Q_1 \preccurlyeq^{k+1} Q_2$, then $Q_2 \xRightarrow{\widehat{\mu}} R$ and $S \preccurlyeq^{k} R$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by Lemma 2.5, we can infer that $Q_2 + P_2 \xRightarrow{\widehat{\mu}} R$ as to be proved (since we already know that $S \preccurlyeq^{k} R$). If $\xRightarrow{\widehat{\mu}}$ is the empty sequence of transitions, then $\mu = \tau$ and $Q_2 = R$. Thus, we have $Q_2 + P_2 \xRightarrow{\widehat{\tau}} Q_2 + P_2$ by the empty sequence of transitions and $S \preccurlyeq^{k} Q_2 + P_2$ by induction (since $S \preccurlyeq^{k} R$ and $Q_2 = R$), as to be proved. The latter case, i.e. $P_1 \xrightarrow{\mu} S$, proceeds symmetrically.

5. The proof proceeds as that of Lemma 4.4, but for using Lemmata 2.6 and 2.7 in place of Lemmata 2.2 and 2.3. Moreover, we have to additionally consider the transitions due to the operational rule *(Close-L)* and its symmetric (in which case the proof proceeds similarly). Thus, suppose the operational rule *(Close-L)* is applied. Then, we have that $Q_1 \mid P_1 \xrightarrow{\tau} (\nu m)(Q_1' \mid P_1')$, for some $m \notin \mathtt{fn}(P_1)$, and, by a shorter proof, that $Q_1 \xrightarrow{\overline{b}(m)} Q_1'$ and $P_1 \xrightarrow{bm} P_1'$. Since, by induction hypothesis, $Q_1 \preccurlyeq^{k+1} Q_2$, we have that $Q_2 \xRightarrow{\overline{b}(m)} Q_2'$ (as $\overline{b}(m)$ is visible then $\xRightarrow{\widehat{\overline{b}(m)}}$ coincides with $\xRightarrow{\overline{b}(m)}$), and $Q_1' \preccurlyeq^{k} Q_2'$. Similarly, since, by induction hypothesis, $P_1 \preccurlyeq^{k+1} P_2$, we have that $P_2 \xRightarrow{bm} P_2'$ and $P_1' \preccurlyeq^{k} P_2'$. Now, by Lemma 2.8, we get that $Q_2 \mid P_2 \xRightarrow{\tau} (\nu m)(Q_2' \mid P_2')$. Moreover, by the induction hypothesis, we get that $(\nu m)(Q_1' \mid P_1') \preccurlyeq^{k} (\nu m)(Q_2' \mid P_2')$, and the thesis is proved.

6. Suppose that $(\nu n)Q_1 \xrightarrow{\mu} Q'$. By the syntactic form of the term, the only applicable operational rules are *(Res)* and *(Open)*, thus, by a shorter proof, we have, respectively

   (a) $Q_1 \xrightarrow{\mu} Q''$, with $n \notin \mathtt{n}(\mu)$ and $Q' = (\nu n)Q''$;

   (b) $\mu = \overline{b}(n)$ and $Q_1 \xrightarrow{\overline{b}n} Q'$ with $n \neq b$.

   In case (a), by induction hypothesis, we have $P_1 \xRightarrow{\widehat{\mu}} P''$ and $Q'' \preccurlyeq^{k} P''$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by applying rule *(Res)*, we obtain $(\nu n)P_1 \xRightarrow{\widehat{\mu}} (\nu n)P''$ and the thesis then follows since, by induction hypothesis, $Q' \preccurlyeq^{k} (\nu n)P''$. Otherwise, $\mu = \tau$ and $P'' = P_1$. By induction, $Q'' \preccurlyeq^{k} P''$ implies $(\nu n)Q'' \preccurlyeq^{k} (\nu n)P'' = (\nu n)P_1$, as to be proved. In case (b), by induction hypothesis, $P_1 \xRightarrow{\overline{b}n} P'$ and $Q' \preccurlyeq^{k} P'$. Then, by applying rule *(Open)*, we obtain $(\nu n)P_1 \xRightarrow{\widehat{\overline{b}(n)}} P'$, as to be proved.

7. Suppose that $!Q_1 \xrightarrow{\mu} Q'$. By the syntactic form of the term, the only applicable operational rules are *(Rep-Act)*, *(Rep-Comm)* and *(Rep-Close)*, thus the only possible transitions are, respectively

   (a) $Q_1 \xrightarrow{\mu} Q''$ and $Q' = Q'' \mid !Q_1$;

(b) $\mu = \tau$, $Q_1 \xrightarrow{\bar{b}m} Q_2$, $Q_1 \xrightarrow{bm} Q_3$ and $Q' = (Q_2 \mid Q_3) \mid !Q_1$;

(c) $\mu = \tau$, $Q_1 \xrightarrow{\bar{b}(m)} Q_2$, $Q_1 \xrightarrow{bm} Q_3$ and $Q' = (\nu m)(Q_2 \mid Q_3) \mid !Q_1$, for some $m \notin \mathtt{fn}(Q_1)$.

In the first case, since by hypothesis $Q_1 \preccurlyeq^{k+1} P_1$, we have that $P_1 \overset{\widehat{\mu}}{\Longrightarrow} P'$ and $Q'' \preccurlyeq^k P'$. If $\overset{\widehat{\mu}}{\Longrightarrow}$ is not the empty sequence of transitions, then, by applying rule *(Rep-Act)*, we obtain $!P_1 \overset{\mu}{\Longrightarrow} P' \mid !P_1$. By induction hypothesis, $!Q_1 \preccurlyeq^k !P_1$, we get that $Q' \preccurlyeq^k P' \mid !P_1$ as required. Otherwise, $\mu = \tau$ and $P' = P_1$. Thus, we have that $Q'' \preccurlyeq^k P_1$. By induction hypothesis, $!Q_1 \preccurlyeq^k !P_1$, we get that $Q' \preccurlyeq^k P_1 \mid !P_1$. Finally, by Lemma 4.6 and by Proposition 4.3, we have that $Q' \preccurlyeq^k !P_1$, as to be proved (since $!P_1 \overset{\widehat{\tau}}{\Longrightarrow} !P_1$). In the second case, since by hypothesis $Q_1 \preccurlyeq^{k+1} P_1$, we have that $P_1 \overset{\bar{b}m}{\Longrightarrow} P_2$ and $Q_2 \preccurlyeq^k P_2$, and that $P_1 \overset{bm}{\Longrightarrow} P_3$ and $Q_3 \preccurlyeq^k P_3$. Now, by applying *(Rep-Comm)*, we get that $!P_1 \overset{\tau}{\Longrightarrow} (P_2 \mid P_3) \mid !P_1$. Finally, by the induction hypothesis, we have that $!Q_1 \preccurlyeq^k !P_1$, hence the thesis follows. In the last case, the proof proceeds as in case (ii), but applying rule *(Rep-Close)*.

□

Although in general input prefix does not preserve $\preccurlyeq^\omega$, we can however prove a narrower result (Lemma 4.9) which suffices for our purposes (Theorem 4.2). We need a couple of preliminary results. We first show that invisible processes are impervious to substitutions, i.e. substitutions application does not cause an invisible process to become visible.

Lemma 4.7 *Let I be an invisible process and σ be a substitution. Then $I\sigma$ is an invisible process.*

Proof. We will prove that, for every process P, substitution σ and visible action β, if $P\sigma \overset{\beta}{\Longrightarrow}$ then $P \overset{\alpha}{\Longrightarrow}$ for some visible action α. From this it follows that, if I is an invisible process, then $I\sigma$ is an invisible process too, otherwise it could be possible to find a visible action α such that $I \overset{\alpha}{\Longrightarrow}$ against the hypothesis. Let $k \geq 0$ be such that $P\sigma \xrightarrow{\tau}_k Q \xrightarrow{\beta}$; we proceed by induction on k.

- If $k = 0$, then $P\sigma \xrightarrow{\beta}$. Thus, if β is $\bar{a}n$ or $\bar{a}(n)$, then, by Lemma 2.4, $P \xrightarrow{\alpha} P'$ with $\alpha\sigma = \beta$. If β is of the form am and m is not fresh, then, by the operational rule *(Inp)* of Table 4, $P\sigma \xrightarrow{an}$ for some n fresh (n exists because $\mathtt{n}(P\sigma) \cap \mathtt{n}(\sigma)$ is finite). Then, again by Lemma 2.4, $P \xrightarrow{\alpha} P'$ with $\alpha\sigma = \beta$.
- If $k > 0$, then $P\sigma \xrightarrow{\tau} P' \overset{\beta}{\Longrightarrow}$. Then, by Lemma 2.4, either $P \xrightarrow{\alpha}$, for some visible action α, or $P \xrightarrow{\tau} P''$ and $P''\sigma = P'$. Since $P' \xrightarrow{\tau}_{k-1} Q \xrightarrow{\beta}$, then, by induction hypothesis, we get that $P'' \overset{\alpha}{\Longrightarrow}$ where α is a visible action, hence we get $P \overset{\alpha}{\Longrightarrow}$.

□

We now show that substitutions application and operators of the calculus safely commute.

Lemma 4.8 *Let C be a context, P a process and σ a substitution such that $\mathtt{bn}(C[P]) \cap \mathtt{n}(\sigma) = \emptyset$. Then $C[P]\sigma = C\sigma[P\sigma]$.*

Proof. The proof proceeds by structural induction on C. The base case is trivial since, when $C = \_$, we have $C[P]\sigma = P\sigma = C\sigma[P\sigma]$. The induction step is a case analysis on the outer operator. To prove the statement, we must show that substitutions application and operators of the calculus safely commute. The most significant cases are when binding operators are taken into account, for which we rely on the hypothesis $\mathtt{bn}(C[P]) \cap \mathtt{n}(\sigma) = \emptyset$ to ensure that σ affects no name bound in $C[P]$. We explicitly show the two cases.

- Let $C = a(x).C'$. Then, we have $C[P]\sigma = (a(x).C'[P])\sigma \overset{(1)}{=} a\sigma(x).C'\sigma[P\sigma] = (a(x).C')\sigma[P\sigma] = C\sigma[P\sigma]$, where (1) holds because by hypothesis $x \notin \mathtt{n}(\sigma)$.

- Let $C = (\nu n)C'$. Then, we have $C[P]\sigma = ((\nu n).C'[P])\sigma \overset{(2)}{=} (\nu n).C'\sigma[P\sigma] = ((\nu n).C')\sigma[P\sigma] = C\sigma[P\sigma]$, where (2) holds because by hypothesis $n \notin \mathtt{n}(\sigma)$.

$\square$

Remark 4.2 *It is worth noticing that contexts are not equated by α-conversion. For example, consider the following two α-convertible contexts $C_1 = (\nu a)\_$ and $C_2 = (\nu b)\_$, and process $P = \bar{a}\,c.\mathbf{0}$; the effect of the two contexts on process P is rather different: process $C_1[P]$ is blocked while $C_2[P]$ is not. In fact, a context can be α-converted as long as the enclosed process is coherently α-converted. Therefore, in the previous Lemma, the hypothesis $\mathtt{bn}(C[P]) \cap \mathtt{n}(\sigma) = \emptyset$ is not restrictive, it simply formalizes our assumption that α-convertible processes are identified and the use of Convention 2.1 on names.*

Remark 4.3 *Lemma 4.8 does not hold for relabelling in place of substitution because the application of relabeling to channels does not act as the substitution of names. Indeed, relabelling preserves $\preccurlyeq^\omega$ (see Proposition 4.6), while substitution does not (see discussion at the beginning of Section 4.2). In fact, substitution is a metanotation, not an operator, and the effect of its application to a term is defined inductively on the syntax of terms. For example, by definition, substitution commutes with parallel composition, while relabelling does not. See e.g. the example at the beginning of Section 4.2 and then consider the variant where $C = \_[a/x]$.*

Although $\preccurlyeq^\omega$ is not a pre-congruence for π-calculus, we can however prove that the premises of Proposition 4.5(1) hold.

Lemma 4.9 *For every context C, invisible process I and process P, it holds that $C[I] \preccurlyeq^\omega C[P]$.*

Proof. The proof proceeds by induction on the structure of contexts. All cases follow from Propositions 4.4 and 4.8, but for input prefix. Now, suppose C is $a(x).C'$. Due to the syntactic form of $a(x).C'[I]$, the only applicable operational rule is *(Inp)* (Table 4). The only transitions are then $a(x).C'[I] \xrightarrow{an} C'[I]\{{}^{n}\!/\!_{x}\}$ for every name n. Similarly, by using the same rule, we get $a(x).C'[P] \xrightarrow{an} C'[P]\{{}^{n}\!/\!_{x}\}$. Now, since α-convertible processes are identified, by possibly renaming a finite number of bound names within I, P and C' we can make so that the hypotheses of Lemma 4.8 hold for both triples I, C', $\{{}^{n}\!/\!_{x}\}$, and P, C', $\{{}^{n}\!/\!_{x}\}$. If I', P' and C'' are the terms obtained from such renaming, and if we let $I'' = I'\{{}^{n}\!/\!_{x}\}$, $P'' = P'\{{}^{n}\!/\!_{x}\}$ and $C''' = C''\{{}^{n}\!/\!_{x}\}$, then, by Lemma 4.8, we have $C[I]\{{}^{n}\!/\!_{x}\} = C'''[I'']$ and $C[P]\{{}^{n}\!/\!_{x}\} = C'''[P'']$. Hence, we get $a(x).C[I] \xrightarrow{an} C'''[I'']$ and $a(x).C[P] \xrightarrow{an} C'''[P'']$. Moreover, by Lemma 4.7, we get that I'' is an invisible process. The thesis then follows since C''' is simpler[1] than $a(x).C'$, thus, by induction, we may assume that $C'''[I_1] \preccurlyeq^\omega C'''[P_1]$, for every invisible process I_1 and process P_1. $\square$

It is now easy to prove that π-calculus is strongly rep-free.

Theorem 4.2 *π-calculus is strongly rep-free.*

Proof. Let C be a context, I be an invisible process, and P be a process. By Lemma 4.9, we have that $C[I] \preccurlyeq^\omega C[P]$. Then, the thesis follows by Proposition 4.5(1). $\square$

[1]In fact, C' and C''' have the same structure, they may only differ for replacing of some names.

5 Weakly replacement free calculi

In this section we deal with some calculi that are only weakly rep-free. Intuitively, for the calculi we consider, violation of strong replacement freeness is due to the introduction of one of three different mechanisms: match among names, polyadic synchronization and pattern matching. Among the many different process calculi that have adopted these mechanisms, for the sake of simplicity we analyze three known variants of π-calculus. For each calculus we show that it is not strongly rep-free by exhibiting a triple made of a context C, an invisible process I and a process P such that condition (2) of Definition 3.2 is violated. In all the exhibited contexts the hole is in the scope of an input prefix: thus, by means of an interaction, the context can generate a substitution whose application to the enclosed invisible process transforms it into a visible process by enabling a visible step that was originally blocked. In fact, for all the calculi we consider in this section, Lemmata 2.4(1) and 4.7 do not hold. Instead, if we restrain invisible processes only to those that are closed, the strategy sketched above does not apply anymore. Finally, to prove that the three calculi are weakly rep-free, instead of three separate proofs, we define a richer variant, called π^{MPM}-calculus, incorporating them all and make the proof for it (Theorem 5.1).

π-calculus with match.

This variant enriches π-calculus with a *match* operator allowing a process to test if two names coincide and to continue its execution only if the test succeeds. This construct has been used in many presentation of π-calculus, as e.g. in the original one [25]. The syntax of the operator is $[x = y]P$, where P is a process and x and y are names. The operational semantics of match is defined by rule *(Mat)* reported in Table 6.

According to our criteria, π-calculus with match operator is strictly more expressive than π-calculus. In fact, we have

Proposition 5.1 *π-calculus with match operator is not strongly rep-free.*

Proof. Let C, I and P be as follows:

$$C = (\nu x)(x(a).\_ \mid \bar{x}b) \qquad I = [a = b]\bar{y}c \qquad P = \mathbf{0}$$

I is an invisible process since it is blocked by an unsatisfied match (as names a and b are supposed to be different). However, we have that $C[I] \Downarrow$, in fact

$$C[I] = (\nu x)(x(a).[a = b]\bar{y}c \mid \bar{x}b) \xrightarrow{\tau} (\nu x)([b = b]\bar{y}c \mid \mathbf{0}) \xrightarrow{\bar{y}c} (\nu x)(\mathbf{0} \mid \mathbf{0})$$

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = (\nu x)(x(a) \mid \bar{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$$

and become an invisible process in doing so. □

Hence, there is no basic encoding from π-calculus with match operator into π-calculus. However, as an immediate consequence of Proposition 5.1 and Theorem 5.1, we have that

Corollary 5.1 *π-calculus with match operator is weakly rep-free.*

Remark 5.1 *Many variants of π-calculus include also a* mismatch *operator that allows a process to test if two names differ before continuing its computation. Differently from match, mismatch does not affect the expressive power of the calculus, that is π-calculus with mismatch is still strongly rep-free and π-calculus with match and mismatch remains weakly rep-free. Intuitively, this is because the application of substitutions to a term does in general increase the number of (free) names that are identified and, consequently, increases the number of successful matches while decreases the number of successful mismatches. Additional computations can then be originated from originally unsuccessful matches that become successful after substitution application, while the opposite holds for mismatch.*

π-calculus with polyadic synchronization.

This variant [10] enriches π-calculus by allowing to use *tuples*, i.e. sequences of names, denoted by $\langle a_1, a_2, \dots, a_n \rangle$ or $\mathbf{a}$, in addition to single names, for identifying input and output channels. Thus, the syntax of input and output actions becomes $\mathbf{a}(x)$ and $\bar{\mathbf{a}}n$, respectively. The operational semantics is defined by rules similar to those of π-calculus where subjects of input and output actions are tuples (see the transition rules in Table 6 defining the semantics of π^{MPM}-calculus, of which π-calculus with polyadic synchronization is a sublanguage).

As a consequence of the following result, π-calculus with polyadic synchronization is strictly more expressive than π-calculus.

Proposition 5.2 *π-calculus with polyadic synchronization is not strongly rep-free.*

Proof. Let C, I and P be as follows:

$$C = (\nu x)(x(a).\_ \mid \bar{x}b) \qquad I = (\nu z)(\overline{\langle z,a\rangle}d \mid \langle z,b\rangle(w).\bar{y}c) \qquad P = \mathbf{0}$$

I is an invisible process since synchronisation along channels $\langle z,a\rangle$ and $\langle z,b\rangle$ cannot take place (as names a and b are supposed to be different). However, we have that $C[I] \Downarrow$, in fact

$$\begin{array}{rcl} C[I] & = & (\nu x)(x(a).(\nu z)(\overline{\langle z,a\rangle}d \mid \langle z,b\rangle(w).\bar{y}c) \mid \bar{x}b) \xrightarrow{\tau} \\ & & (\nu x)((\nu z)(\overline{\langle z,b\rangle}d \mid \langle z,b\rangle(w).\bar{y}c) \mid \mathbf{0}) \xrightarrow{\tau} \\ & & (\nu x)((\nu z)(\mathbf{0} \mid \bar{y}c) \mid \mathbf{0}) \xrightarrow{\bar{y}c} \\ & & (\nu x)((\nu z)(\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}) \end{array}$$

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = (\nu x)(x(a) \mid \bar{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$$

and become an invisible process in doing so. $\square$

Hence, there is no basic encoding from π-calculus with polyadic synchronization into π-calculus. However, as an immediate consequence of Proposition 5.2 and Theorem 5.1, we have that

Corollary 5.2 *π-calculus with polyadic synchronization is weakly rep-free.*

π-calculus with pattern matching.

This variant enriches π-calculus by allowing input actions to use *patterns of names* to selectively synchronise with output actions along the same channel according to the offered tuples. Pattern matching was introduced in the context of π-calculus in [18], but it is also used by several other process calculi, like e.g. those presented in [21, 23]. A pattern is a sequence of names where some names are placeholders while some other ones are not and stand for themselves. We indicate these latter ones by operator $\ulcorner\cdot\urcorner$. Intuitively, a pattern $\mathbf{x}$ can match any tuple $\mathbf{a}$ having the same length obtained by instantiating the names in $\mathtt{fn}(\mathbf{x})$, where any name occurring in $\mathbf{x}$ is free but for those that are argument of operator $\ulcorner\cdot\urcorner$, thus, e.g., $\ulcorner x\urcorner\sigma = \ulcorner x\urcorner$ for any name x and substitution σ. To check if a pattern $\mathbf{x}$ and a tuple $\mathbf{a}$ match, we define a partial function $match(\cdot,\cdot)$ that, in the positive case, returns the least substitution σ such that $\mathbf{x}\sigma = \mathbf{a}$ (once the occurrences of operator $\ulcorner\cdot\urcorner$ in the left hand side have been removed). The function is defined inductively on the syntax of its two arguments by the following clauses:

$$match(\langle\rangle,\langle\rangle) = \varepsilon \qquad match(x,a) = \{a/x\} \qquad match(\ulcorner a\urcorner,a) = \varepsilon$$
$$match(\langle x,\mathbf{x}\rangle,\langle a,\mathbf{a}\rangle) = \{a/x\}\circ match(\mathbf{x},\mathbf{a}) \quad match(\langle\ulcorner a\urcorner,\mathbf{x}\rangle,\langle a,\mathbf{a}\rangle) = match(\mathbf{x},\mathbf{a})$$

where $\langle\rangle$ denotes the empty pattern/tuple, ε the identity function, $\circ$ substitution composition, and $\langle b,\mathbf{a}\rangle$ denotes the sequence obtained by concatenating the name b with the sequence $\mathbf{a}$. Thus, a process $a(\mathbf{x}).P$ and a process $\overline{a}\mathbf{b}.Q$ can synchronise if, and only if, $match(\mathbf{x},\mathbf{b}) = \sigma$, for some σ, and, after the synchronisation, $a(\mathbf{x}).P$ becomes $P\sigma$. The operational semantics is defined by rules similar to those of π-calculus where objects of input and output actions are replaced by patterns and tuples, respectively, and rule *(Inp)* calls function $match(\cdot,\cdot)$ to check possible matching tuples (see the transition rules reported in Table 6 defining the semantics of π^{MPM}-calculus, of which π-calculus with pattern matching is a sublanguage).

According to our criteria, π-calculus with pattern matching is strictly more expressive than π-calculus. In fact,

Proposition 5.3 *π-calculus with pattern matching is not strongly rep-free.*

Proof. Let C, I and P be as follows:

$$C = (\nu x)(x(a).\_ \mid \bar{x}b) \qquad I = (\nu z)(\bar{z}a \mid z(\ulcorner b\urcorner).\bar{y}c) \qquad P = \mathbf{0}$$

I is an invisible process since it is blocked because the pattern $\ulcorner b\urcorner$ and the tuple a do not match (as names a and b are supposed to be different). However, we have that $C[I]\Downarrow$, in fact

$$\begin{array}{rcl} C[I] & = & (\nu x)(x(a).(\nu z)(\bar{z}a \mid z(\ulcorner b\urcorner).\bar{y}c) \mid \bar{x}b) \xrightarrow{\tau} \\ & & (\nu x)((\nu z)(\bar{z}b \mid z(\ulcorner b\urcorner).\bar{y}c) \mid \mathbf{0}) \xrightarrow{\tau} \\ & & (\nu x)((\nu z)(\mathbf{0} \mid \bar{y}c) \mid \mathbf{0}) \xrightarrow{\bar{y}c} \\ & & (\nu x)((\nu z)(\mathbf{0} \mid \mathbf{0}) \mid \mathbf{0}) \end{array}$$

Instead, $C[P]\not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = (\nu x)(x(a) \mid \bar{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$$

and become an invisible process in doing so. □

Therefore, there is no basic encoding from π-calculus with pattern matching into π-calculus. However, as an immediate consequence of Proposition 5.3 and Theorem 5.1, we have that

Corollary 5.3 *π-calculus with pattern matching is weakly rep-free.*

$$\begin{array}{rcl} P,Q & ::= & \mathbf{0} \;\mid\; \mu.P \;\mid\; P+Q \;\mid\; P|Q \;\mid\; (\nu n)P \;\mid\; !P \;\mid\; [n=m]P \;\mid\; [n\neq m]P \\ \mu & ::= & \tau \;\mid\; \mathbf{a}(\mathbf{x}) \;\mid\; \bar{\mathbf{a}}\,\mathbf{n} \end{array}$$

Table 5: π^{MPM}-calculus syntax

$$\frac{}{\bar{\mathbf{a}}\mathbf{n}.P \xrightarrow{\bar{\mathbf{a}}\mathbf{n}} P}\;\textit{(Out)} \qquad \frac{match(\mathbf{x},\mathbf{n})=\sigma}{\mathbf{a}(\mathbf{x}).P \xrightarrow{\mathbf{a}\mathbf{n}} P\sigma}\;\textit{(Inp)} \qquad \frac{P \xrightarrow{(\mathbf{y})\bar{\mathbf{a}}\mathbf{n}} P'}{(\nu x)P \xrightarrow{(\langle x,\mathbf{y}\rangle)\bar{\mathbf{a}}\mathbf{n}} P'}\; x\in\mathbf{n}, x\notin\mathbf{a}\;\textit{(Open)}$$

$$\frac{P \xrightarrow{(\mathbf{y})\bar{\mathbf{a}}\mathbf{n}} P' \quad Q \xrightarrow{\mathbf{a}\mathbf{n}} Q'}{P \mid Q \xrightarrow{\tau} (\nu\mathbf{y})(P' \mid Q')}\; \mathbf{y}\cap\mathtt{fn}(Q)=\emptyset\;\textit{(Close-L)}$$

$$\frac{P \xrightarrow{(\mathbf{y})\bar{\mathbf{a}}\mathbf{n}} P' \quad P \xrightarrow{\mathbf{a}\mathbf{n}} P''}{!P \xrightarrow{\tau} (\nu\mathbf{y})(P' \mid P'')\mid !P}\; \mathbf{y}\cap\mathtt{fn}(P)=\emptyset\;\textit{(Rep-Close)}$$

$$\frac{P \xrightarrow{\mu} P'}{[n=n]P \xrightarrow{\mu} P'}\;\textit{(Mat)} \qquad \frac{P \xrightarrow{\mu} P'}{[n\neq m]P \xrightarrow{\mu} P'}\; n\neq m\;\textit{(Mismat)}$$

Table 6: (Excerpt of) Labelled transition rules of π^{MPM}-calculus

π^{MPM}-calculus.

The three variants of π-calculus previously presented can be combined to form a sort of super calculus, that we call π^{MPM}-calculus, corresponding to simultaneously adding match, mismatch, polyadic synchronization and pattern matching to π-calculus. The syntax of π^{MPM}-calculus is reported in Table 5, while the most significant labelled transition rules defining the operational semantics of π^{MPM}-calculus are reported in Table 6. We have omitted rules *(Tau)*, *(Ch-L)*, *(Res)*, *(Par-L)* and *(Rep-Act)*, since they are the same as those of π-calculus in Table 4, and the symmetric forms of rules *(Ch-L)*, *(Comm-L)*, *(Close-L)* and *(Par-L)*. With respect to π-calculus some slight adaptations have been made concerning communication actions. Basically, subjects are tuples while objects are tuples/patterns and, when inputs are performed, patterns and tuples are required to match. The labels of the transition rules are generated by the grammar

$$\mu \quad ::= \quad \mathbf{a}\,\mathbf{n} \quad\mid\quad (\mathbf{m})\bar{\mathbf{a}}\,\mathbf{n} \quad\mid\quad \tau$$

where $(\mathbf{m})\bar{\mathbf{a}}\,\mathbf{n}$ indicates output via $\mathbf{a}$ of the tuple $\mathbf{n}$ with exported bound names $\mathbf{m}$.

We now prove that π^{MPM}-calculus is weakly rep-free from which it follows that all the three variants of π-calculus we have considered in this section are weakly rep-free too. We will use Lemma 4.8, that can be easily extended to π^{MPM}-calculus, and the following result.

Lemma 5.1 *Let P be a process and n and m be names with $n\neq m$. Then $P \preccurlyeq^{\omega} [n=n]P$ and $P \preccurlyeq^{\omega} [n\neq m]P$.*

Proof. We prove by induction on k that $P \preccurlyeq^{k} [n=n]P$ ($P \preccurlyeq^{k} [n\neq m]P$ can be proved similarly). The base case trivially follows by definition of $\preccurlyeq^{0}$ (Definition 4.1). To prove the induction step, we assume the statement true for k and prove that it also holds for $k+1$. Suppose that $P \xrightarrow{\mu} Q$. Now, by applying the

operational rule *(Mat)*, we get that $[n=n]P \xrightarrow{\mu} Q$ and the thesis follows from reflexivity (Lemma 4.1).
□

Theorem 5.1 *π^{MPM}-calculus is weakly rep-free.*

Proof. We show that $C[I] \preccurlyeq^{\omega} C[P]$, for every context C, closed invisible process I and process P. The thesis then follows by Proposition 4.5(2). Similarly to that of Lemma 4.9, the proof is an easy induction on the structure of context C but replacing respectively:

- subjects and objects of input and output actions with tuples and patterns;
- invisible processes with closed invisible processes;
- Lemma 4.7 with Proposition 3.2.

We only explicitly consider here the cases of operators match and input (where pattern matching is used). The proof for operator mismatch is similar to that of operator match, while the remaining cases are similar to those of Proposition 4.8. For each operator, we prove by induction on k that $\preccurlyeq^k$ is preserved. The base case trivially follows by definition of $\preccurlyeq^0$ (Definition 4.1). To prove the induction step, we assume the statement true for k and prove that it also holds for $k+1$.

- Suppose that $[n=m]C[I] \xrightarrow{\mu} Q$. Due to the syntactic form of the term, the only applicable operational rule is *(Mat)* (Table 6), thus we have $n=m$ and $C[I] \xrightarrow{\mu} Q$ by a shorter proof. By induction hypothesis, $C[P] \xRightarrow{\widehat{\mu}} P'$ and $Q \preccurlyeq^k P'$. If $\xRightarrow{\widehat{\mu}}$ is not the empty sequence of transitions, then, by applying rule *(Mat)*, we obtain $[n=n]C[P] \xRightarrow{\widehat{\mu}} P'$, which implies the thesis. Otherwise, $\mu = \tau$ and $P' = C[P]$ and, by induction, $Q \preccurlyeq^k C[P]$. By Lemma 5.1, $C[P] \preccurlyeq^{\omega} [n=n]C[P]$, thus, by Proposition 4.3, we get that $Q \preccurlyeq^k [n=n]C[P]$, as to be proved.
- Suppose that $\mathbf{a}(\mathbf{x}).C[I] \xrightarrow{\mu} Q$. Due to the syntactic form of $\mathbf{a}(\mathbf{x}).C[I]$, the only applicable operational rule is *(Inp)* (Table 6). The only transitions are then $\mathbf{a}(\mathbf{x}).C[I] \xrightarrow{\mathbf{an}} C[I]\sigma$ with $match(\mathbf{x},\mathbf{n}) = \sigma$. Similarly, by using the same rule, we get $\mathbf{a}(\mathbf{x}).C[P] \xrightarrow{\mathbf{an}} C[P]\sigma$. Now, since α-convertible processes are identified, by possibly renaming a finite number of bound names within I, P and C, we can make so that the hypotheses of Lemma 4.8 hold for both triples I, C, σ, and P, C, σ. If I', P' and C' are the terms obtained from such renaming, and if we let $P'' = P'\sigma$ and $C'' = C'\sigma$, then, by Lemma 4.8, we have $C[I]\sigma = C''[I']$ (in fact, since I is closed, then I' is closed too thus $I' = I'\sigma$ and, by Proposition 3.2, I' is invisible) and $C[P]\sigma = C''[P'']$. Hence, we get $\mathbf{a}(\mathbf{x}).C[I] \xrightarrow{\mathbf{an}} C''[I']$ and $\mathbf{a}(\mathbf{x}).C[P] \xrightarrow{\mathbf{an}} C''[P'']$. The thesis then follows since C'' is simpler[2] than $\mathbf{a}(\mathbf{x}).C$, thus by induction on the structure of contexts, we may assume that $C''[I_1] \preccurlyeq^{\omega} C''[P_1]$, for every process P_1 and invisible process I_1.

□

6 Non replacement free calculi

In this section we show that most of the calculi with some form of priority proposed in the literature, as e.g. those presented in [1, 32, 9, 13, 23, 7], are not rep-free. In process calculi, priority is one of the most widely studied, and natural, notions used to implement different levels of urgency between actions of (a system of) processes. According to the terminology of [13] (that surveys the different

[2]In fact, C and C'' have the same structure, they only differ for replacing of some names.

$$P \ ::= \ \mathbf{0} \ \mid \ \mu.P \ \mid \ P+P \ \mid \ \Theta(P)$$

Table 7: BCCSP$_\Theta$ syntax

$$\mu.P \xrightarrow{\mu} P \qquad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'} \qquad \frac{Q \xrightarrow{\mu} Q'}{P+Q \xrightarrow{\mu} Q'} \qquad \frac{P \xrightarrow{\mu} P' \quad \neg(P \xrightarrow{\mu'} \text{ for } \mu < \mu')}{\Theta(P) \xrightarrow{\mu} \Theta(P')}$$

Table 8: Labelled transition rules of BCCSP$_\Theta$

approaches taken in the literature), we consider both calculi with local priority (as e.g. CPG [32], CCS with priority choice [9], and COWS [23]), and calculi with global priority (as e.g. BCCSP$_\Theta$ [1], CCS^{sg} and CCS$^{\mathsf{prio}}$ [13]).

The results presented in this section demonstrate that there exists no independence preserving basic encoding from any of the calculi with priority into, e.g., CCS or π-calculus, or into any of the calculi we have presented in Section 5. For each calculus we show that it is not rep-free by exhibiting a triple made of a context C, a closed invisible process I (usually the null process $\mathbf{0}$) and a process P such that condition (3) of Definition 3.4 is violated.

For the sake of simplicity, the fragments of the calculi presented in this section will be slightly adapted and simplified to avoid, as much as possible, introducing further notations and complications.

BCCSP$_\Theta$.

BCCSP$_\Theta$ (BCCSP with the priority operator Θ, [1]) is the simpler calculus with priority analyzed in this paper. It is obtained by adding the well-known priority operator Θ of [4] to the basic process algebra BCCSP [37].

In Table 7 we report an utterly simplified syntax of BCCSP$_\Theta$ that, unlike the original one [1], does not allow action and process variables. The priority operator Θ gives certain actions priority over others based on an irreflexive partial ordering relation $<$ over the set of actions. Intuitively, $\mu < \mu'$ is interpreted as 'μ' has priority over μ'. Thus, for example, if P is some process that can initially perform both μ and μ', then $\Theta(P)$ will initially only be able to execute μ'. That is, in the context of the priority operator Θ, action μ is pre-empted by action μ'.

The transition rules are reported in Table 8. All the rules are standard but for the last one that captures the operational intuition underlying the priority operator. It exploits the *negative* premise '$\neg(P \xrightarrow{\mu'}$ for $\mu < \mu')$' to state that a process $\Theta(P)$ can initially perform a μ-labelled transition if its argument P has such a transition and cannot initially perform any action μ' that has higher priority than μ.

Proposition 6.1 *BCCSP$_\Theta$ is not rep-free.*

Proof. Let C, I and P be as follows:

$$C = \Theta(a + \_) \qquad I = \mathbf{0} \qquad P = \tau$$

with $a < \tau$. We have that $C[I] \Downarrow$, in fact $C[I] = \Theta(a+\mathbf{0})$ can perform the transition

$$\Theta(a+\mathbf{0}) \xrightarrow{a} \Theta(\mathbf{0})$$

$$P \ ::= \ \mathbf{0} \ \mid \ \textstyle\sum_{i\in I} S_i : \mu_i.P_i \ \mid \ P[f] \ \mid \ new\ a\ P \ \mid \ P \mid Q \ \mid \ A\langle a_1,\ldots,a_n\rangle$$

Table 9: CPG syntax

$$\frac{\mu_i \notin S_i}{\sum_{i\in I} S_i : \mu_i.P_i \xrightarrow{S_i:\mu_i} P_i} \qquad \frac{P \xrightarrow{S:\mu} P'}{new\ a\ P \xrightarrow{S\setminus\{a,\overline{a}\}:\mu} new\ a\ P'}\ \mu \notin \{a,\overline{a}\}$$

$$\frac{P \xrightarrow{S:\mu} P' \quad Q\ eschew\ S}{P \mid Q \xrightarrow{S:\mu} P' \mid Q} \qquad \frac{P \xrightarrow{S_1:\mu} P' \quad Q \xrightarrow{S_2:\overline{\mu}} Q' \quad P\ eschew\ S_2 \quad Q\ eschew\ S_1}{P \mid Q \xrightarrow{(S_1\cup S_2):\tau} P' \mid Q'}$$

Table 10: (Excerpt of) Labelled transition rules of CPG

Instead, $C[P] \not\Downarrow$ since process $C[P] = \Theta(a+\tau)$ can only perform the transition

$$\Theta(a+\tau) \xrightarrow{\tau} \Theta(\mathbf{0})$$

and become an invisible process in doing so. □

It is worth noticing that even such an extremely simple calculus with priority cannot be properly encoded into any rep-free calculus.

CPG.

CPG (CCS with Priority Guards, [32]) is an extension of CCS allowing processes with *priority guards*, i.e. terms of the form $S : \mu.P$, where S is some finite set of visible actions (i.e. $S \subset \mathcal{N} \cup \overline{\mathcal{N}}$), which behave like $\mu.P$ except that action μ can only be performed if the environment does not offer any action in $\overline{S}$. In Table 9, we report the syntax of the calculus. CPG builds on a variant of CCS where the choices are guarded, restrictions are done one name at time and parameterized process definitions are used in place of recursion for modelling infinite behaviours. Thus, the term *new a P* is semantically equivalent to $P\backslash\{a\}$, while $A\langle a_1,\ldots,a_n\rangle$ is the invocation of process identifier A with parameters $a_1,\ldots,a_n$ (for every process identifier B, it is assumed that there exists one defining equation of the form $B(b_1,\ldots,b_n) \overset{\text{def}}{=} P$).

In CPG, a transition can be conditional on offers from the environment. Consider $\{a\} : b \mid \overline{b}$. The process can make a computation step due to the synchronization between b and $\overline{b}$. However b is guarded by a, and so the computation step is conditional on the environment not offering $\overline{a}$. This is reflected by letting transitions be parameterised on labels of the form $S : \mu$. The intended meaning of $P \xrightarrow{S:\mu} P'$ is that P can perform action μ, and become P' in doing so, as long as the environment *eschews* S, i.e. does not offer $\overline{\alpha}$ for any $\alpha \in S$. The transition rules relevant for our purposes are reported in Table 10.

Proposition 6.2 *CPG is not rep-free.*

Proof. Let C, I and P be as follows:

$$C = new\ a\ (new\ b\ ((a + \{a\} : b.\overline{c}) \mid \overline{b} \mid \_)) \qquad I = \mathbf{0} \qquad P = \overline{a}$$

We have that $C[I] \Downarrow$, in fact

$$\begin{array}{rcl} C[I] & = & new\ a\ (new\ b\ ((a + \{a\} : b.\overline{c}) \mid \overline{b} \mid \mathbf{0})) \xrightarrow{\{a\}:\tau} \\ & & new\ a\ (new\ b\ (\overline{c} \mid \mathbf{0} \mid \mathbf{0})) \xrightarrow{\emptyset:\overline{c}} \\ & & new\ a\ (new\ b\ (\mathbf{0} \mid \mathbf{0} \mid \mathbf{0})) \end{array}$$

where $\emptyset : \overline{c}$ is visible while $\{a\} : \tau$ is not. Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = new\ a\ (new\ b\ ((a + \{a\} : b.\overline{c}) \mid \overline{b} \mid \overline{a})) \xrightarrow{\emptyset:\tau} new\ a\ (new\ b\ (\mathbf{0} \mid \overline{b} \mid \mathbf{0}))$$

labelled by the invisible action $\emptyset : \tau$ and become an invisible process in doing so. □

The semantics of the fragment of CPG where the only allowed priority guarded processes are of the form $\emptyset : \mu.P$ coincides with that (of a variant) of CCS. Hence, CPG is somehow more expressive than CCS. In [32, 39] it is also argued that expressiveness of CPG and π-calculus is not comparable.

A less flexible prioritised variant of CCS is introduced in [9], where priority can be decided in a way which is specific to each choice in a term. In this case priorities are statically fixed according to the structure of the term. The prioritised choice, written $P +\rangle\ Q$, allows Q to make a transition only if P cannot perform actions for synchronising with the environment. The transition relation is of the form $P \xrightarrow{\mu}_R Q$ and means that, in an environment which is ready to perform precisely the output actions R, process P can perform an action μ to become Q. The operational rules for prioritised choice are:

$$\frac{P \xrightarrow{\mu}_R P'}{P +\rangle\ Q \xrightarrow{\mu}_R P'} \qquad \frac{Q \xrightarrow{\mu}_R Q' \quad \neg(P \xrightarrow{\mu'}_R P' \wedge \mu' \in \overline{R} \cup \{\tau\})}{P +\rangle\ Q \xrightarrow{\mu}_R Q'}$$

We show that also this calculus is not rep-free by slightly modifying the terms we used in Proposition 6.2. Thus, let C, I and P be as follows:

$$C = ((a +\rangle\ b.\overline{c}) \mid \overline{b} \mid \_)\backslash\{a,b\} \qquad I = \mathbf{0} \qquad P = \overline{a}$$

We have that $C[I] \Downarrow$, in fact

$$\begin{array}{rcl} C[I] & = & ((a +\rangle\ b.\overline{c}) \mid \overline{b} \mid \mathbf{0})\backslash\{a,b\} \xrightarrow{\tau}_{\emptyset} \\ & & (\overline{c} \mid \mathbf{0} \mid \mathbf{0})\backslash\{a,b\} \xrightarrow{\overline{c}}_{\emptyset} \\ & & (\mathbf{0} \mid \mathbf{0} \mid \mathbf{0})\backslash\{a,b\} \end{array}$$

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = ((a +\rangle\ b.\overline{c}) \mid \overline{b} \mid \overline{a})\backslash\{a,b\} \xrightarrow{\tau}_{\emptyset} (\mathbf{0} \mid \overline{b} \mid \mathbf{0})\backslash\{a,b\}$$

labelled by the invisible action $\emptyset : \tau$ and become an invisible process in doing so.

$$
\begin{array}{ccc}
\dfrac{}{\underline{\mu}.P \xrightarrow{\underline{\mu}} P} &
\dfrac{P \xrightarrow{\underline{\mu}} P'}{P + Q \xrightarrow{\underline{\mu}} P'} &
\dfrac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}\ \underline{\tau} \notin \underline{I}(Q) \\[2ex]
 & \dfrac{P \xrightarrow{\underline{\mu}} P'}{P \mid Q \xrightarrow{\underline{\mu}} P' \mid Q} &
\dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \underline{\tau} \notin \underline{I}(P \mid Q) \\[2ex]
 & \dfrac{P \xrightarrow{\underline{\alpha}} P' \quad Q \xrightarrow{\underline{\overline{\alpha}}} Q'}{P \mid Q \xrightarrow{\underline{\tau}} P' \mid Q'} &
\dfrac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \underline{\tau} \notin \underline{I}(P \mid Q)
\end{array}
$$

Table 11: (Excerpt of) Labelled transition rules of CCS^{sg}

CCS^{sg} and CCS$^{\mathsf{prio}}$.

CCS^{sg} (CCS with static priority and global pre-emption, [13]) is an extension of CCS where channels have priority levels and only complementary actions at the same level of priority can engage in a communication. A notion of pre-emption then stipulates that a process cannot engage in transitions labelled by actions with a given priority whenever it is able to perform a transition labelled by an internal action of a higher priority. In this case, we say that the lower-priority transition is pre-empted by the higher-priority internal transition. Therefore, visible actions never have pre-emptive power over actions of lower priority because visible actions only indicate the potential for execution.

For simplicity, we consider just two priority levels: ordinary actions and higher priority, underlined actions. The syntax of CCS^{sg} is then the same as that of CCS reported in Table 1, except for actions that can also be underlined.

The transition rules relevant for our purposes are reported in Table 11. In the presentation of the operational rules, notation $\underline{I}(P)$ denotes the set of all prioritized actions in which process P can initially engage. When restricting only to unprioritized (or only to prioritized) actions, the transition rules of CCS^{sg} are exactly the ones of CCS reported in Table 2. When both prioritized and unprioritized actions may be involved, an unprioritized action is allowed only if no prioritized invisible action can be executed.

CCS$^{\mathsf{prio}}$ [13] extends CCS^{sg} with two operators, originally introduced in [12], which correspond to the prioritization of a visible unprioritized action, written $P\lceil\mu$, and to the deprioritization of a visible prioritized action, written $P\lfloor\underline{\mu}$.

Proposition 6.3 *CCS^{sg} and CCS$^{\mathsf{prio}}$ are not rep-free.*

Proof. We first prove the statement for CCS^{sg}. Let C, I and P be as follows:

$$C = (\underline{a} \mid \_)\backslash\{\underline{a}\} + \overline{b} \qquad I = \mathbf{0} \qquad P = \underline{\overline{a}}$$

We have that $C[I] \Downarrow$, in fact

$$C[I] = (\underline{a} \mid \mathbf{0})\backslash\{\underline{a}\} + \overline{b} \xrightarrow{\overline{b}} \mathbf{0}$$

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = (\underline{a} \mid \underline{\overline{a}})\backslash\{\underline{a}\} + \overline{b} \xrightarrow{\underline{\tau}} (\mathbf{0} \mid \mathbf{0})\backslash\{\underline{a}\}$$

$$P,Q \quad ::= \quad \mathbf{0} \quad | \quad \mathbf{kill}(\kappa) \quad | \quad \bar{\mathbf{a}}\,\mathbf{n} \quad | \quad \mathbf{a}(\mathbf{x}).P \quad | \quad [\kappa]P \quad | \quad P\,|\,Q$$

Table 12: COWS syntax

and become an invisible process in doing so.

To prove the statement for CCS$^{\mathsf{prio}}$ it is sufficient to take

$$C = ((\underline{a} \mid \_)\backslash\{\underline{a}\} + \overline{b})\lceil b\,.$$

The rest of the proof proceeds similarly. □

Proposition 6.3 allows to conclude that there exists no basic encoding of CCS^{sg} into CCS. Since the fragment of CCS^{sg} not containing prioritized actions coincides with CCS, and the identity encoding is a basic encoding of CCS into CCS^{sg}, we obtain that CCS^{sg} is strictly more expressive than CCS.

COWS.

COWS (Calculus for Orchestration of Web Services, [23]) is a recent formalism specifically devised for modelling service-oriented computing (SOC) systems which integrates primitives of well-known process calculi (e.g. π-calculus) with constructs meant to model web services orchestration (e.g. communication endpoints and forced termination). In COWS, service definitions and service instances are both modelled as reactive processes running concurrently and two different kinds of priority mechanisms, combining dynamic priority with local pre-emption [13], are used for services orchestration purposes. One mechanism assigns input actions priority values which depend on the output tuples available along the same channel so that, if more patterns match the same tuple, the inputs using a more defined pattern have greater priority. This way, service instances take precedence over the corresponding service definition when both can process the same message, thus preventing creation of wrong new instances. The other mechanism assigns actions for forcing immediate termination of concurrent processes greatest priority within their enclosing scope. This way, when a fault arises in a scope, (some of) the remaining processes of the enclosing scope can be terminated before starting the execution of the relative fault handler (and similarly for exception and compensation handling).

Due to space limitations, we consider here only a very simple fragment of the original calculus (without replication/choice/protection operators) and borrow many notations from π^{MPM}-calculus (see Section 5). Table 12 shows this syntax. In addition to the set of names $\mathscr{N}$, we assume existence of a disjoint set of *killer labels* $\mathscr{K}$, ranged over by κ and κ'. They can be used for introducing a named scope for grouping certain processes. It is then possible to associate suitable termination activities to such a scope, as well as ad hoc fault and compensation handlers, thus laying the foundation for guaranteeing transactional properties in spite of services' loose coupling. Being different from names, killer labels cannot be exchanged in communications, thus their scope is statically regulated by the *delimitation* operator.

The transition rules relevant for our purposes are reported in Table 13. In practice we focus on the two novel operators, namely kill and delimitation. Activity $\mathbf{kill}(\kappa)$ causes immediate termination of all concurrent activities inside an enclosing $[\kappa]$, that stops the killing effect by turning the transition label κ into τ. Execution of parallel processes is interleaved, but when a kill activity can be performed. In fact, kill activities are executed *eagerly* with respect to the activities enclosed within the delimitation of the corresponding killer label. Differently from the scope of a restricted name in π-calculus, the scope of a delimited killer label is statically fixed.

$$\mathbf{kill}(\kappa) \xrightarrow{\kappa} \mathbf{0} \qquad \frac{P \xrightarrow{\kappa} P'}{P \mid Q \xrightarrow{\kappa} P'} \qquad \frac{P \xrightarrow{\mu} P' \qquad \mu \neq \kappa}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathtt{bn}(\mu) \cap \mathtt{fn}(Q) = \emptyset$$

$$\frac{P \xrightarrow{\kappa} P'}{[\kappa]P \xrightarrow{\tau} [\kappa]P'} \qquad \frac{P \xrightarrow{\mu} P' \quad \neg(P \xrightarrow{\kappa})}{[\kappa]P \xrightarrow{\mu} [\kappa]P'}$$

Table 13: (Excerpt of) Labelled transition rules of COWS

Proposition 6.4 COWS *is not rep-free.*

Proof. Let C, I and P be as follows:

$$C = [\kappa]\,(\,\_\ \mid \bar{\mathbf{a}}\mathbf{n}\,) \qquad I = \mathbf{0} \qquad P = \mathbf{kill}(\kappa)$$

We have that $C[I] \Downarrow$, in fact

$$C[I] = [\kappa]\,(\,\mathbf{0} \mid \bar{\mathbf{a}}\mathbf{n}\,) \xrightarrow{\bar{\mathbf{a}}\mathbf{n}} [\kappa]\,(\,\mathbf{0} \mid \mathbf{0}\,)$$

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition

$$C[P] = [\kappa]\,(\,\mathbf{kill}(\kappa) \mid \bar{\mathbf{a}}\mathbf{n}\,) \xrightarrow{\tau} [\kappa]\,(\,\mathbf{0} \mid \mathbf{0}\,)$$

and become an invisible process in doing so. □

BIP.

The BIP (Behavior, Interaction, Priority [7]) component framework is a formalism for modeling heterogeneous component-based systems. It allows the description of systems as the composition of generic atomic components characterised by their behavior and their interfaces. BIP offers two powerful mechanisms for describing composition of components by combining interactions and priorities. Interactions are used to specify multiparty synchronization between components, while priorities between interactions are used to restrict non determinism inherent to parallel systems.

A system model has three layers. The lowest layer contains atomic components. An atomic component B_i is defined by a labelled transition system $\langle \mathscr{P}_i, \mathscr{L}_i, \longrightarrow \rangle$. The set $\mathscr{L}$ of all labels in the system is the union of the sets $\mathscr{L}_i$ of labels of its atomic components which are assumed to be pairwise disjoint.

The second layer describes possible interactions between atomic components. An interaction is a finite set $\mu = \{\mu_i\}_{i \in I}$ (with $I \subseteq \{1, \ldots, n\}$) of labels. A set Λ of interactions characterises a (composite) component $B \overset{\text{def}}{=} \Lambda(B_1, \ldots, B_n)$ in terms of a labelled transition system whose states are a combination $\langle Q_1, \ldots, Q_n \rangle$ of the states of the atomic components and whose transition relation is given by the least set of transitions induced by the rule

$$\frac{\mu = \{\mu_i\}_{i \in I} \in \Lambda \quad \forall i \in I : Q_i \xrightarrow{\mu_i} Q_i' \quad \forall i \notin I : Q_i = Q_i'}{\langle Q_1, \ldots, Q_n \rangle \xrightarrow{\mu} \langle Q_1', \ldots, Q_n' \rangle}$$

Intuitively, the inference rule says that a composite component can execute an interaction $\mu \in \Lambda$ iff for each label $\mu_i \in \mu$, the corresponding atomic component can execute the transition labelled with μ_i; the states of components that do not participate in the interaction stay unchanged.

The third layer specifies priorities between interactions. A priority relation is a strict partial order $<_Q$ among interactions parameterised by a state Q. Thus, $\mu' <_Q \mu''$ means that, at state Q, interaction μ' has less priority than iteration μ''. Given a component $\langle \mathscr{P}, \mathscr{L}, \longrightarrow \rangle$ and some priority relations $<_Q$ for $Q \in \mathscr{P}$, by exploiting the rule

$$\frac{Q \xrightarrow{\mu} Q' \quad \forall \mu' \in \mathscr{L} : \mu <_Q \mu' \Rightarrow \neg(Q \xrightarrow{\mu'})}{Q \xrightarrow{\mu}_{<} Q'}$$

we can construct a new component $\langle \mathscr{P}, \mathscr{L}, \longrightarrow_{<} \rangle$ where, among all the possible interactions, only those with higher priority are allowed. Notably, the above rule can be rewritten into a semantically equivalent one following the style of the rule for operator Θ in Table 8:

$$\frac{Q \xrightarrow{\mu} Q' \quad \neg(Q \xrightarrow{\mu'} \text{ for } \mu <_Q \mu')}{Q \xrightarrow{\mu}_{<} Q'}$$

BIP is quite expressive and permits to model many typical process calculi operators possibly dealing with priority. In particular, it is easy to model BCCSP$_\Theta$ and specifically its Θ operator (just take, for any state Q, $<_Q = <$ where $<$ is the partial ordering relation used by Θ) from which, due to Proposition 6.1, it immediately follows that

Corollary 6.1 *BIP is not rep-free.*

7 Concluding remarks and related work

To sum up, a first contribution of this paper is the introduction of a metatheory based on the replacement freeness criterion and on the notion of basic (possibly, independence preserving) encodings that provides a method to tell process calculi apart. Our approach is general and uniform enough to permit comparing the relative expressive power of quite different process calculi. On the contrary, many works on the subject only focus on variants of the same calculus. A second contribution consists in presenting a number of results coming from the application of our metatheory to well-known process calculi, that are possibly extensions of CCS or π-calculus. We thus end up to retrieve separation results similar to, e.g., [19, 18], but ours are stronger since they hold for a more general class of encodings, or to, e.g., [10, 32, 39], but here they follow by possibly simpler proofs. Finally, some other results, e.g. some of those for non rep-free calculi presented in Section 6, as far as we know, are pointed out for the first time.

Related work

Different approaches have been proposed in the literature for analysing the expressive power of the great variety of process calculi developed in the last decades. We refer the interested reader to [31] for a survey on the subject. More traditional methods try to assess the absolute expressive power of a calculus by studying implementability of some Turing complete formalism or decidability of certain properties. These methods are however not much discriminating since most of the calculi are Turing complete, i.e. they can compute the same class of functions as Turing machines. Thus, unlike computability theory, the capability of two calculi of simulating Turing machines does not imply equality in their expressiveness. Hence, other ways of proceeding have been investigated mainly focussing on comparing the relative expressive power of more calculi.

One method consists in studying which problems can be solvable in a calculus under some conditions that cannot be met by any solution in another calculus. A calculus is considered more expressive than another one, if it provides solutions to more problems. For example, several variants of CCS, π-calculus and Ambient calculus [11] have been compared by means of the 'leader election' problem [15, 30, 33, 40, 32, 34, 39]. Another method is to investigate which operators are definable in a given calculus and not in another one (see e.g. [14, 5, 36]). However, the identification of a problem solvable, as well as that of an operator expressible, in a calculus but not in another one is in general difficult.

A completely different approach for studying the relative expressive power of process calculi consists in either defining an encoding of one in the other and analysing the properties of the encoding function, or in proving that such an encoding cannot exist. This is an effective approach that aims at comparing the calculi just on the basis of their semantic differences, i.e. without resorting to any specific problem. However, the significance of the obtained results is subordinated to the properties that the encodings are required to enjoy, i.e. to the classes of encodings. Our work belongs to this trend of research (see e.g. [13, 10, 30, 19, 18, 32, 3, 39, 16]) and is close to [18], that introduces the class of *reasonable encodings* for comparing several communication primitives in the context of π-calculus, and [19], that introduces the class of *valid encodings* for comparing many variants of CCS, π-calculus and Ambient calculus [11]. Our basic encodings generalise the reasonable encodings, the former ones being obtained from the latter ones by dropping the conditions on name invariance, operational correspondence and divergence preservation and reflection. Similarly, our basic encodings generalise the valid encodings by dropping the conditions on name invariance, operational correspondence and divergence reflection. Indeed, *success sensitiveness* of [19] implies our interaction sensitiveness requirement: appropriate observers can be defined that test the capability of a process to interact with the environment and report success only in that case. In fact, [18, 19] aim at identifying suitable criteria for both encodability and separation results, thus the considered encodings are the outcome of a compromise between 'maximality' (typical of encodability results) and 'minimality' (typical of separation results), while we are only interested to separation results and the weaker the requirements on the encodings, the stronger the results. Moreover, we additionally consider process calculi with priority, thus we end up establishing different separation results.

In fact, we impose somewhat coarser demands on our encodings than those usually found in the literature. For example, we don't require operational correspondence (as instead done in e.g. [38, 18, 19]), observational correspondence or full abstraction (as instead done in e.g. [17, 29, 16]), and in place of homomorphism (of e.g. parallel composition as required in [10, 30, 38, 34]) we simply require compositionality. In particular, the requirement of homomorphy for parallel composition entails that the encoding respects the distribution of a term into parallel components exactly, i.e. without introducing any sort of coordinating context that would reduce the degree of distribution. This requirement has been sometimes criticized and indeed there exist encodings that do not translate parallel composition homomorphically (see e.g. [27, 6]). Moreover, it is quite strong when compared to the requirement of compositionality, imposed to ours basic encodings, that only implies that any context in the source calculus can be represented as a context in the target calculus. Compositionality is a very natural property and, indeed, every encoding we are aware of is defined compositionally.

Distribution preservation, i.e. homomorphy for parallel composition, has been used as a requirement (for the encodings) in [32, 39] for separating CPG from both CCS and π-calculus, thus obtaining results similar to our Proposition 6.2. It is a requirement for the class of *uniform encodings* (that, other than distribution, are required to preserve renaming, i.e. to respect permutation of free names) introduced in [30] for comparing the expressive power of synchronous and asynchronous versions of the π-calculus, which instead our criteria do not allow to separate. In fact, with a proof similar to, but simpler than, that of Theorem 4.2, we can show that asynchronous π-calculus [2] is rep-free as well. An even stronger class of encodings is used in [10] to establish some separation results for variants of π-calculus. Among these results, the authors prove that match and polyadic synchronization cannot be encoded in π-calculus, which are similar to the separation results we present in Section 5. Distribution preservation is also required for the encodings considered in [34], where expressiveness of different variants of Ambient calculus is analysed, and in [38], where an extension of the π-calculus with both polyadic synchronization and priority is introduced and used as target of some 'reasonable' encodings of some bio-inspired process calculi.

For some of the calculi with priority considered in this paper, there are already some separation results with respect to calculi without priority (see, e.g., [39, 32]). However, these results are given by considering either a less expressive fragment of CCS as the target calculus or a strict class of encodings, typically uniform encodings. In fact, distribution preservation seems even more restrictive when calculi with priority mechanisms are involved. Indeed, priority alters the very basic notion of distributed computation, since in calculi with priority it is possible to know in advance if a process is not ready to perform some synchronisation [39], which is instead not decidable in calculi without priority. It seems then too demanding to require that the parallel operator of a calculus with priority exactly maps to the corresponding operator of a calculus without priority.

In [3] the authors consider some syntactic variants of CCS with replication in place of recursion and study the expressiveness of restriction and its interplay with replication. They also enrich one of these variants with priority guards [32] and conclude that priority adds expressivity to this variant of the calculus. As comparison criteria they consider decidability of convergence and relative expressiveness with respect to a well-known observational semantics, i.e. *failure semantics*. Differently, we consider CCS and its extension CPG and do not rely on any observational semantics.

Future work

We intend to apply our metatheory to more process calculi, as e.g. the Extended pi-Calculi [22] and the Psi-calculi [8], that have already turn out to be quite expressive, and the prioritised variant of π-calculus introduced in [38]. Other more challenging applications, that might require an appropriate tuning of our metatheory, concern process calculi with communication mechanisms different from those considered in this paper, as e.g. the broadcast variant of π-calculus considered in [40], Ambient calculus and higher order π-calculus [35]. Moreover, we also plan to investigate the impact of loosening the requirement of preserving name independence. Indeed, although the requirement is used in many classes of encodings (as e.g. those used in [30, 40, 32, 39]), it leaves out of our study all those encodings that exploit some kind of *reserved* names for rendering specific primitives and operators of the source calculus. In this case, the encodings of independent processes might end up not to be independent. Examples of such encodings can be found, e.g., in [18, 20]. This may also lead to weakening the demand of compositionality for allowing the encoding of a process of the source language to be defined by combining the encodings of its subprocesses through a single outermost context that coordinates their inter-relationships.

Acknowledgements We thank Daniele Gorla and the anonymous reviewers of EXPRESS'10 for their fruitful comments that have helped us in improving the paper.

References

[1] L. Aceto, T. Chen, W. Fokkink & A. Ingólfsdóttir (2008): *On the axiomatisability of priority*. Mathematical Structures in Computer Science 18(1), pp. 5–28.

[2] R.M. Amadio, I. Castellani & D. Sangiorgi (1998): *On Bisimulations for the Asynchronous pi-Calculus*. Theor. Comput. Sci. 195(2), pp. 291–324.

[3] J. Aranda, F.D. Valencia & C. Versari (2009): *On the Expressive Power of Restriction and Priorities in CCS with Replication*. In: FOSSACS, LNCS 5504. Springer, pp. 242–256.

[4] J. Baeten, J. Bergstra & J.W. Klop (1986): *Syntax and defining equations for an interrupt mechanism in process algebra*. Fundamenta Informaticae IX(2), pp. 127–168.

[5] J.C.M. Baeten, J.A. Bergstra & J.W. Klop (1987): *On the Consistency of Koomen's Fair Abstraction Rule*. Theor. Comput. Sci. 51, pp. 129–176.

[6] M. Baldamus, J. Parrow & B. Victor (2005): *A Fully Abstract Encoding of the π-calculus with Data Terms*. In: ICALP, LNCS 3580. Springer, pp. 1202–1213.

[7] A. Basu, P. Bidinger, M. Bozga & J. Sifakis (2008): *Distributed Semantics and Implementation for Systems with Interaction and Priority*. In: FORTE. Springer, pp. 116–133.

[8] J. Bengtson, M. Johansson, J. Parrow & B. Victor (2009): *Psi-calculi: Mobile Processes, Nominal Data, and Logic*. In: LICS. IEEE Computer Society Press, pp. 39–48.

[9] J. Camilleri & G. Winskel (1995): *CCS with priority choice*. Inf. Comput. 116(1), pp. 26–37.

[10] M. Carbone & S. Maffeis (2003): *On the expressive power of polyadic synchronisation in π-calculus*. Nordic Journal of Computing 10(2), pp. 70–98.

[11] L. Cardelli & A.D. Gordon (2000): *Mobile ambients*. Theor. Comput. Sci. 240(1), pp. 177–213.

[12] R. Cleaveland & M. Hennessy (1990): *Priorities in process algebras*. Inf. Comput. 87(1-2), pp. 58 – 77.

[13] R. Cleaveland, G. Lüttgen & V. Natarajan (2001): *Priorities in process algebras*. Handbook of Process Algebra, chapter 12 , pp. 711–765.

[14] R. de Simone (1985): *Higher-Level Synchronising Devices in Meije-SCCS*. Theor. Comput. Sci. 37, pp. 245–267.

[15] C. Ene & T. Muntean (1999): *Expressiveness of Point-to-Point versus Broadcast Communications*. In: Fundamentals of Computation Theory (FCT'99), LNCS 1684. Springer, pp. 258–268.

[16] Y. Fu & H. Lu (2010): *On the expressiveness of interaction*. Theor. Comput. Sci. 441(11-13), pp. 1387–1451.

[17] P. Giambagi, G. Schneider & F.D. Valencia (2004): *On the Expressive of Infinite Behaviour and Name Scoping in Process Calculi*. In: FOSSACS, LNCS 2987. Springer, pp. 226–240.

[18] D. Gorla (2008): *Comparing communication primitives via their relative expressive power*. Inf. Comput. 206(8), pp. 931–952.

[19] D. Gorla (2008): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. In: CONCUR, LNCS 5201. Springer, pp. 492–507.

[20] D. Gorla (2009): *On the Relative Expressive Power of Calculi for Mobility*. ENTCS 249, pp. 269–286.

[21] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi & G. Zavattaro (2006): *SOCK: A Calculus for Service Oriented Computing*. In: ICSOC, LNCS 4294. Springer, pp. 327–338.

[22] M. Johansson, J. Parrow, B. Victor & J. Bengtson (2008): *Extended pi-Calculi*. In: ICALP, LNCS 5126. Springer, pp. 87–98.

[23] A. Lapadula, R. Pugliese & F. Tiezzi (2007): *A Calculus for Orchestration of Web Services.* In: ESOP, LNCS 4421. Springer, pp. 33–47.

[24] R. Milner (1989): *Communication and concurrency.* Prentice-Hall.

[25] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, I and II*. Inf. Comput. 100(1), pp. 1–40, 41–77.

[26] R. Milner & D. Sangiorgi (1992): *Barbed Bisimulation*. In: Proc. of 19th International Colloquium on

Automata, Languages and Programming (ICALP), *LNCS* 623. Springer, pp. 685–695.

[27] U. Nestmann (2000): *What is a "Good" Encoding of Guarded Choice?* *Inf. Comput.* 156(1-2), pp. 287–319.

[28] U. Nestmann (2006): *Welcome to the Jungle: A Subjective Guide to Mobile Process Calculi*. In: *CONCUR*, *LNCS* 4137. Springer, pp. 52–63.

[29] U. Nestmann & B.C. Pierce (2000): *Decoding Choice Encodings*. *Inf. Comput.* 163(1), pp. 1–59.

[30] C. Palamidessi (2003): *Comparing the expressive power of the synchronous and asynchronous π-calculi*. *Mathematical Structures in Computer Science* 13(5), pp. 685–719.

[31] J. Parrow (2008): *Expressiveness of Process Algebras*. In: *LIX*, *ENTCS* 209. Elsevier Science, pp. 173–186.

[32] I. Phillips (2008): *CCS with priority guards*. *Logic and Algebraic Programming* 75(1), pp. 139–165.

[33] I. Phillips & M.G. Vigliotti (2006): *Leader election in rings of ambient processes*. *Theor. Comput. Sci.* 356(3), pp. 468–494.

[34] I. Phillips & M.G. Vigliotti (2008): *Symmetric electoral systems for ambient calculi*. *Inf. Comput.* 206(1), pp. 34–72.

[35] D. Sangiorgi & D. Walker (2001): *The π-calculus: A Theory of Mobile Processes*. Cambridge Univ. Press.

[36] F.W. Vaandrager (1992): *Expressive Results for Process Algebras*. In: *Sematics: Foundations and Applications (REX Workshop)*, *LNCS* 666. Springer, pp. 609–638.

[37] R.J. van Glabbeek (1990): *The Linear Time-Branching Time Spectrum*. In: *CONCUR*, *LNCS* 458. Springer, pp. 278–297.

[38] C. Versari (2007): *A Core Calculus for a Comparative Analysis of Bio-inspired Calculi*. In: *ESOP*, *LNCS* 4421. Springer, pp. 411–425.

[39] C. Versari, N. Busi & R. Gorrieri (2009): *An expressiveness study of priority in process calculi*. *Mathematical Structures in Computer Science* 19(6), pp. 1161–1189.

[40] M.G. Vigliotti, I. Phillips & C. Palamidessi (2007): *Tutorial on separation results in process calculi via leader election problems*. *Theor. Comput. Sci.* 388(1-3), pp. 267–289.