A criterion for separating process calculi

Federico Banti Rosario Pugliese Francesco Tiezzi
Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy
fbanti@gmail.com 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.

1 Introduction

The field of process calculi has been sometimes compared to a 'jungle of interrelated but separate theories' [24], 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., [11, 8, 26, 16, 15, 27, 28, 3, 34, 13]), 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 en-*

Submitted to Electronic Proceedings
in Theoretical Computer Science.

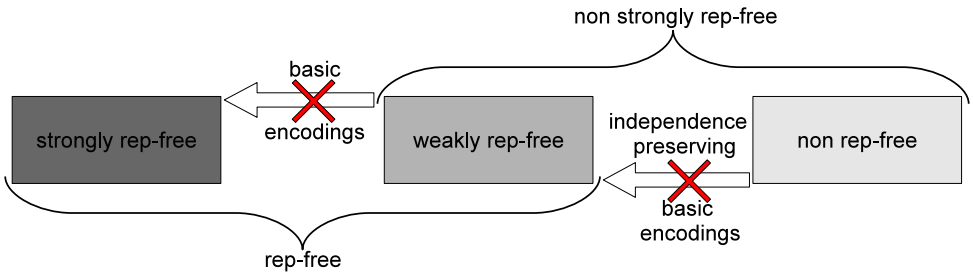

Figure 1: Calculi partition

codings, introduced in [15] 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 independence* [26, 29] (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 [21] and π-calculus [22], 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, 28, 11, 20], 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 [22], polyadic synchronization [8] or pattern matching [15] 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 [28, 8, 16, 15, 34]. 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 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 3 proves that CCS and π-calculus are strongly rep-free, Section 4 proves that some variants of π-calculus are only weakly rep-free, and Section 5 proves that several calculi with priority are not (even weakly) rep-free. Finally, Section 6 draws a few conclusions and reviews some strictly related work. To save space, we only outline the techniques used in the proofs and omit some calculi from our presentation; we refer the interested reader to [6] for their full account and for a wider comparison with related work.

2 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 next sections, we present several results about the relative expressiveness of well-known process calculi arising from the exploitation of our methodology.

2.1 Background notations

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 ranged over by $\mu, \mu', \mu_1, \ldots$ and 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). Specifically, actions are expressed in terms of *names*, i.e. basic entities without structure, ranged over by letters a, b, $\ldots$, x, y, $\ldots$, n, m, $\ldots$. 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 $\mathtt{fn}(P)$ for the set of names that have a free occurrence in P. A process P is *closed* if $\mathtt{fn}(P) = \emptyset$.

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 2.1 (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* $\__1, \ldots, \__k$*. 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.

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 $\mathbb{C}$.

We assume that the operational semantics of process calculi is defined by means of *labelled transition systems*. Transitions labelled by invisible actions correspond to *computation steps* and can be thought of as taking place of 'internal' interactions of systems, whereas transitions labelled by visible actions can be thought of as representing only 'potential' computation steps, since in order for them to occur they require a contribution from the environment. As usual, we will write $P \xrightarrow{\mu} P'$ to indicate that the process P can do a transition labelled μ and become the process P' in doing so. We let $\Longrightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$, $\stackrel{\mu}{\Longrightarrow}$ to denote $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ (the juxtaposition of two relations indicates their composition), and $\stackrel{\widehat{\mu}}{\Longrightarrow}$ to denote $\Longrightarrow$, if $\mu = \tau$, and $\stackrel{\mu}{\Longrightarrow}$, otherwise. Moreover, we will write $\xrightarrow{\mu}_k$ to denote the composition of $\xrightarrow{\mu}$ with itself k-times (similarly for the other transition relations).

Now, by exploiting the relations above, we define the following predicates over processes.

Definition 2.2 (Process predicates) *Let* P *be a process.*

- $P \Downarrow_\alpha$*, i.e.* P can perform the (visible) action α*, if* $P \stackrel{\alpha}{\Longrightarrow} P'$ *for some* P'*;*
- $P \Downarrow$*, i.e.* P *is* visible*, if there exists a visible action* α *such that* $P \Downarrow_\alpha$*;*
- $P \not\Downarrow$*, i.e.* P *is* invisible*, if there exists no visible action* α *such that* $P \Downarrow_\alpha$*.*

Predicate $\Downarrow_\alpha$ accounts for the ability of processes of interacting with their environment. Notably, an invisible process either is stuck or can only perform invisible actions. Clearly, if P is invisible, then it can only evolve to invisible processes.

Remark 2.1 *Our results are equally true for process calculi whose operational semantics is defined by means of* reduction relations*, instead of LTSs. In this case, predicate* $\Downarrow_\alpha$ *is defined by induction on the syntax of processes.*

2.2 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 2.3 (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* $\forall\, P_1, \ldots, P_k \in \mathbb{C}_1$*,* $[\![op(P_1, \ldots, P_k)]\!] = C_{op}[[\![P_1]\!], \ldots, [\![P_k]\!]]$*.*
- $[\![\cdot]\!]$ *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 2.1 *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 this property for 1-hole contexts only, but it could be easily generalised.

Lemma 2.1 *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 $P \in \mathbb{C}_1$, $[\![C_1[P]]\!] = C_2[[\![P]\!]]$.*

Proof (sketch). By induction on the structure of C_1. □

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 2.4 (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 $\mathbb{C}$,*

$$C[I] \Downarrow \qquad \textit{implies} \qquad C[P] \Downarrow \tag{1}$$

A calculus is not strongly rep-free if there exists a triple C, I and P violating the condition of Definition 2.4. 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 2.1 *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 2.1, $[\![I]\!]$ is invisible. By Lemma 2.1, 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 2.1, $C'[[\![I]\!]] \Downarrow$ and $C'[[\![P]\!]] \not\Downarrow$ against the initial assumption that $\mathbb{C}_2$ is strongly rep-free. □

2.3 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 (i.e. a function on names). As shown in Section 4, 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 5. 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, i.e. without free names, and hence 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 4 turn out to be (weakly) rep-free, but not strongly rep-free. Instead, the process calculi with priority presented in Section 5 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 [26, 29], i.e. guarantee that if two processes do not share free names, the same holds for their encodings.

Definition 2.5 (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 that if P is a closed process, then $P\sigma = P$, for any substitution σ. Similarly, 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 2.2 *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 2.5), 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. □

The weak version of replacement freeness is defined by imposing condition (1) on triples C, I and P, where I is a closed invisible process.

Definition 2.6 (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 $\mathbb{C}$,*

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

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 2.1 for rep-free calculi by further requiring the basic encodings to be independence preserving.

Theorem 2.2 *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 2.1 but exploiting the hypothesis that I is closed and that the property of being closed is invariant under independence preserving encodings (Lemma 2.2). □

3 Proving strong replacement freeness of mainstream calculi

In this section we prove that the two well-known process calculi 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 3.1 will then allow us to conclude.

Definition 3.1 (ω-simulation)

1. *$\preccurlyeq^0$ is the universal relation on processes.*
2. *For $0 \leq k < \omega$, $\preccurlyeq^k$ is defined by:*
   $Q \preccurlyeq^k P$ if, whenever $Q \xrightarrow{\mu} Q'$, then, for some P', $P \xRightarrow{\widehat{\mu}} P'$ 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.*

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.

Remark 3.1 *The previous relations resemble the 'stratification' of weak bisimulation for π-calculus [31, 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 [21, 31] about the stratification of weak bisimulation, we can also presume that, in CCS and π-calculus, $\preccurlyeq^\omega$ coincides with the standard* simulation preorder.

The following proposition states that $C[I] \preccurlyeq^\omega C[P]$ implies conditions (1) of Definition 2.4 and (2) of Definition 2.6. It then provides a technique for proving (strongly) replacement freeness, not an alternative characterization. In fact, 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 only requires that $C[P]$ is able to perform some visible action.

Proposition 3.1 (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 $C[I] \Downarrow_\alpha$ 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 3.1 we get that $C[P] \Longrightarrow_m R' \stackrel{\alpha}{\Longrightarrow} R''$ for some processes R' and R''. Hence, $C[P] \Downarrow$ and the thesis is proved. □

CCS is strongly replacement free. To prove that CCS is strongly rep-free (Theorem 3.1), by Proposition 3.1(1), it suffices to show that $\preccurlyeq^\omega$ is a pre-congruence for CCS.

Proposition 3.2 *Relation $\preccurlyeq^\omega$ is a pre-congruence for CCS.*

Proof (sketch). First, for each operator of finite CCS, it is proven by induction on k that $\preccurlyeq^k$ is preserved for every k. Then, it is proven that $\preccurlyeq^\omega$ is preserved by recursive definitions by following the same strategy adopted in [21] for proving that strong bisimulation is preserved by recursive definitions. □

Theorem 3.1 *CCS is strongly rep-free.*

Proof. Let C be a context, I be an invisible process, and P be a process. Since $I \preccurlyeq^\omega P$, by Proposition 3.2, we get that $C[I] \preccurlyeq^\omega C[P]$. Then, the thesis follows by Proposition 3.1(1). □

π-calculus is strongly replacement free. To prove that π-calculus is a strongly rep-free calculus (Theorem 3.2), 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. This happens because, intuitively, execution of an input action can generate a substitution that identifies names that were originally different. As a consequence of the application of this substitution to the continuation process, new transitions could arise because communications that were originally impossible become enabled, thus the expansion of parallel composition in terms of choice and prefix becomes unsound.

Another difference with the previous section is the proof technique used. Indeed, to prove that $\preccurlyeq^\omega$ is preserved by all operators but input prefix, we exploit the fact that our ω-simulation is strictly weaker than a well-known observational semantics for π-calculus, named *weak bisimilarity* [22, 31] and denoted by $\approx$. This fact can be easily proved by relying on a family of relations $\approx_k$ [31, Def. 2.4.24, page 99], that stratify the definition of weak bisimilarity and whose intersection $\approx_\omega$ is weaker than $\approx$ [31, Theorem 2.4.27, page 100] but stronger than $\preccurlyeq^\omega$. Indeed, for every k, we have $\approx_k \subseteq \preccurlyeq^k$ (it directly follows by definition since $\approx_k$ is the greatest symmetric relation contained in $\preccurlyeq^k$) and this implies that $\approx_\omega \subseteq \preccurlyeq^\omega$. Therefore, for proving the following results about $\preccurlyeq^\omega$ we can exploit all the laws that are sound for $\approx$.

Although in general input prefix does not preserve $\preccurlyeq^\omega$, we can however prove, by induction on the structure of contexts, that the premises of Proposition 3.1(1) hold (Lemma 3.1) which suffices for our purposes (Theorem 3.2).

Lemma 3.1 *For every context C, invisible process I and process P, it holds that $C[I] \preccurlyeq^\omega C[P]$.*

Theorem 3.2 *π-calculus is strongly rep-free.*

Proof. Let C be a context, I be an invisible process, and P be a process. By Lemma 3.1, we have that $C[I] \preccurlyeq^\omega C[P]$. Then, the thesis follows by Proposition 3.1(1). □

4 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 (1) of Definition 2.4 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. Instead, if we restrain invisible processes only to those that are closed, the strategy sketched above does not apply anymore. 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 4.1).

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 [22]. The syntax of the operator is $[x=y]P$, where P is a process and x and y are names.

Proposition 4.1 *π-calculus with match operator is not strongly rep-free.*

Proof. Let C, I and P be as follows: $C = (\nu x)(x(a).\_ \mid \overline{x}b)$, $I = [a=b]\overline{y}c$ and $P = \mathbf{0}$. Process I is invisible 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 $C[I] = (\nu x)(x(a).[a=b]\overline{y}c \mid \overline{x}b) \xrightarrow{\tau} (\nu x)([b=b]\overline{y}c \mid \mathbf{0}) \xrightarrow{\overline{y}c} (\nu x)(\mathbf{0} \mid \mathbf{0})$. Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition $C[P] = (\nu x)(x(a) \mid \overline{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$ and become an invisible process in doing so. □

π-calculus with polyadic synchronization. This variant [8] enriches π-calculus by allowing to use *tuples*, i.e. sequences of names, denoted by $\langle a_1, a_2, \ldots, a_n \rangle$ 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 $\overline{\mathbf{a}}n$, respectively. The operational semantics is defined by rules similar to those of π-calculus where subjects of input and output actions are tuples, allowing interaction to happen only when such tuples match.

Proposition 4.2 *π-calculus with polyadic synchronization is not strongly rep-free.*

Proof. Let C, I and P be as follows: $C = (\nu x)(x(a).\_ \mid \overline{x}b)$, $I = (\nu z)(\overline{\langle z,a \rangle} d \mid \langle z,b \rangle (w).\overline{y}c)$ and $P = \mathbf{0}$. Process I is invisible 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

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

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition $C[P] = (\nu x)(x(a) \mid \overline{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$ and become an invisible process in doing so. □

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 [15], but it is also used by several other process calculi, like e.g. those presented in [18, 20]. 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. When the check if a pattern $\mathbf{x}$ and a tuple $\mathbf{a}$ match succeeds, it 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). Thus, a process $a(\mathbf{x}).P$ and a process $\overline{a}\mathbf{b}.Q$ can synchronise if, and only if, $\mathbf{x}$ and $\mathbf{b}$ match by generating some substitution 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 the rule for input actions checks possible matching tuples.

Proposition 4.3 *π-calculus with pattern matching is not strongly rep-free.*

Proof. Let C, I and P be as follows: $C = (\nu x)(x(a).\_ \mid \overline{x}b)$, $I = (\nu z)(\overline{z}a \mid z(\ulcorner b \urcorner).\overline{y}c)$ and $P = \mathbf{0}$. Process I is invisible 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

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

Instead, $C[P] \not\Downarrow$ since process $C[P]$ can only perform the transition $C[P] = (\nu x)(x(a) \mid \overline{x}b) \xrightarrow{\tau} (\nu x)(\mathbf{0} \mid \mathbf{0})$ and become an invisible process in doing so. □

π^{MPM}-calculus. As a consequence of Propositions 4.1, 4.2 and 4.3, we have that the three variants of π-calculus previously presented are strictly more expressive than π-calculus, in other words there is no basic encoding from any of them into π-calculus. Now, we combine them to form a sort of super calculus, that we call π^{MPM}-calculus, corresponding to simultaneously adding match, polyadic synchronization and pattern matching to π-calculus. The syntax of π^{MPM}-calculus is defined as

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

To save space, the presentation of the operational semantics of π^{MPM}-calculus is relegated to [6].

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.

Theorem 4.1 *π^{MPM}-calculus is weakly rep-free.*

Proof (sketch). 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 3.1(2). Similarly to that of Lemma 3.1, 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, and by exploiting the fact that if I is a closed invisible process, then $I\sigma$ is a closed invisible process too. □

5 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, 28, 11, 20], 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 [11] (that surveys the different approaches taken in the literature), we consider both calculi with local priority (as e.g. CPG [28] and COWS [20]), and calculi with global priority (as e.g. BCCSP_Θ [1], CCS^{sg} and CCS^{prio} [11]).

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 extensions of π-calculus we have presented in Section 4. 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 (2) of Definition 2.6 is violated.

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

BCCSP_Θ. BCCSP_Θ (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 [32]. An utterly simplified syntax of BCCSP_Θ that, unlike the original one [1], does not allow action and process variables, is as follows

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

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'.

Proposition 5.1 *$BCCSP_\Theta$ is not rep-free.*

Proof. Let C, I and P be as follows: $C = \Theta(a + \_)$, $I = \mathbf{0}$ and $P = \tau$, with $a < \tau$. We have that $C[I] \Downarrow$, in fact $C[I] = \Theta(a+\mathbf{0})$ can perform the transition $\Theta(a+\mathbf{0}) \xrightarrow{a} \Theta(\mathbf{0})$. 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, [28]) 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, which behave like $\mu.P$ except that action μ can only be performed if the environment does not offer any action in $\overline{S}=\{\overline{n}\mid n\in S\}$. The syntax of the calculus is as follows

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

CPG builds on a variant of CCS where the choices are guarded and parameterized process definitions are used in place of recursion for modelling infinite behaviours.

A transition can be conditional on offers from the environment. Consider $\{a\}:b\mid\overline{b}$. It 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 does not offer $\overline{\alpha}$ for any $\alpha\in S$.

Proposition 5.2 *CPG is not rep-free.*

Proof. Let C, I and P be as follows: $C=new\ a\ (new\ b\ ((a+\{a\}:b.\overline{c})\mid\overline{b}\mid\_))$, $I=\mathbf{0}$ and $P=\overline{a}$. We have that $C[I]\Downarrow$, in fact $C[I]\ =\ new\ a\ (new\ b\ ((a+\{a\}:b.\overline{c})\mid\overline{b}\mid\mathbf{0}))\xrightarrow{\{a\}:\tau} new\ a\ (new\ b\ (\overline{c}\mid\ \mathbf{0}\mid\mathbf{0}))\xrightarrow{\emptyset:\overline{c}} new\ a\ (new\ b\ (\mathbf{0}\mid\ \mathbf{0}\mid\mathbf{0}))$, 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 [28, 34] it is also argued that expressiveness of CPG and π-calculus is not comparable.

CCS^{sg} and CCS$^{\mathsf{prio}}$. CCS^{sg} (CCS with static priority and global pre-emption, [11]) 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, except for actions that can also be underlined.

When restricting only to unprioritized (or only to prioritized) actions, the transition rules of CCS^{sg} are exactly the ones of CCS. When both prioritized and unprioritized actions may be involved, an unprioritized action is allowed only if no prioritized invisible action can be executed.

CCS$^{\mathsf{prio}}$ [11] extends CCS^{sg} with two operators, originally introduced in [10], which correspond to the prioritization of a visible unprioritized action, written $P\lceil\mu$, and to the deprioritisation of a visible prioritized action, written $P\lfloor\underline{\mu}$.

Proposition 5.3 *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: $C=(\underline{a}\mid\_)\backslash\{\underline{a}\}+\overline{b}$, $I=\mathbf{0}$ and $P=\overline{\underline{a}}$. 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 $C[P]=(\underline{a}\mid\overline{\underline{a}})\backslash\{\underline{a}\}+\overline{b}\xrightarrow{\tau}(\mathbf{0}\mid\mathbf{0})\backslash\{\underline{a}\}$ 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 5.3 permits 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, [20]) is a recent formalism specifically devised for modelling service-oriented 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). COWS is equipped with a priority mechanism that 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.

Due to space limitations, we consider here only the very simple fragment of the original calculus (without replication/choice/protection operators) generated by the following syntax

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

In addition to the set of names, we assume existence of a disjoint set of *killer labels*, ranged over by κ. They can be used for introducing a named scope for grouping certain processes. Being different from names, killer labels cannot be exchanged in communications, thus their scope is statically regulated by the *delimitation* operator $[\kappa]P$.

We comment on the two novel operators, namely kill and delimitation. $\mathbf{kill}(\kappa)$ causes immediate termination of all concurrent processes 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 can be performed. In fact, kill is executed *eagerly* with respect to the processes enclosed within the delimitation of the corresponding killer label.

Proposition 5.4 COWS *is not rep-free.*

Proof. Let C, I and P be as follows: $C = [\kappa](\_ \mid \bar{\mathbf{a}}\,\mathbf{n})$, $I = \mathbf{0}$ and $P = \mathbf{kill}(\kappa)$. 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. □

6 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., [16, 15], but ours are stronger since they hold for a more general class of encodings, or to, e.g., [8, 28, 34], 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 5, as far as we know, are pointed out for the first time.

Related work. Defining an encoding of a process calculus into another one and analysing the properties of the encoding function, or proving that such an encoding cannot exist, is a widely adopted approach for studying the relative expressive power of process calculi. 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 (e.g. the 'leader election' problem considered in [12, 26, 29, 35, 28, 30, 34]). 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.

The most closely related works are [15], that introduces the class of *reasonable encodings* for comparing several communication primitives in the context of π-calculus, and [16], that introduces the class of *valid encodings* for comparing many variants of CCS, π-calculus and Ambient calculus [9]. 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 [16] 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, [15, 16] 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. [33, 15, 16]), observational correspondence or full abstraction (as instead done in e.g. [14, 25, 13]), and in place of homomorphism (of e.g. parallel composition as required in [8, 26, 33, 30]) 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. [23, 5]). 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 [28, 34] for separating CPG from both CCS and π-calculus, thus obtaining results similar to our Proposition 5.2. 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 [26] 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 3.2, we can show that asynchronous π-calculus [2] is rep-free as well. An even stronger class of encodings is used in [8] 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 4. Distribution preservation is also required for the encodings considered in [30], where expressiveness of different variants of Ambient calculus is analysed, and in [33], 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., [34, 28]). However, these results are given by considering either a less expressive fragment of CCS as the target calculus or a strict class of encod-

ings, 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 [34], 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 [28] 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. For space limitation, in this paper we considered a subset of the calculi dealt with in [6]. We intend to apply our metatheory to more process calculi, as e.g. the Extended pi-Calculi [19] and the Psi-calculi [7], that have already turn out to be quite expressive, and the prioritised variant of π-calculus introduced in [33]. 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 [35], Ambient calculus and higher order π-calculus [31]. 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 [26, 35, 28, 34]), 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 [15, 17]. 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 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] 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.

[6] F. Banti, R. Pugliese & F. Tiezzi (2010): *A criterion for separating process calculi*. Technical Report, DSI, Università di Firenze. `http://rap.dsi.unifi.it/cows/papers/repFreeFull.pdf`.

[7] 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.

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

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

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

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

[12] 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.

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

[14] 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.

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

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

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

[18] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[34] 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.

[35] 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.