Responsiveness in process calculi*

Lucia Acciai[1] and Michele Boreale[2]

[1] Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence.
[2] Dipartimento di Sistemi e Informatica, Università di Firenze.
lucia.acciai@lif.univ-mrs.fr, boreale@dsi.unifi.it

Abstract. In a process calculus, an agent guarantees responsive usage of a channel name r if a communication along r is guaranteed to eventually take place. Responsiveness is important, for instance, to ensure that any request to a service be eventually replied. We propose two distinct type systems, each of which statically guarantees responsive usage of names in well-typed pi-calculus processes. In the first system, we achieve responsiveness by combining techniques for deadlock and livelock avoidance with *linearity* and *receptiveness*. The latter is a guarantee that a name is ready to receive as soon as it is created. These conditions imply relevant limitations on the nesting of actions and on multiple use of names in processes. In the second system, we relax these requirements so as to permit certain forms of nested inputs and multiple outputs. We demonstrate the expressive power of the two systems by showing that primitive recursive functions – in the case of the first system – and Cook and Misra's service orchestration language ORC – in the case of the second system – can be encoded into well-typed processes.

1 Introduction

In a process calculus, an agent guarantees responsive usage of a channel name r if a communication along r is guaranteed to eventually take place. That is, under a suitable assumption of fairness, all computations contain at least one reduction with r as subject. We christen this property *responsiveness* as we are particularly interested in the case where r is a return channel passed to a service or function. As an example, a network of processes S may contain a service $!a(x,r).P$ invocable in RPC style: the caller sends at a an argument x and a return channel r. S's responsive usage of r implies that every request at a will be eventually replied. This may be a critical property in domains of applications such as service-oriented computing.

Our goal is to individuate substantial classes of pi-calculus processes that guarantee responsiveness and that can be statically checkable. In the past decade, several type systems for the pi-calculus have been proposed to analyze properties that share some similarities with responsiveness, such as linearity [9], uniform receptiveness [12], lock freedom [5,6] and termination [4]; they will be examined throughout the paper. However none of the above mentioned properties alone is sufficient, or even necessary, to ensure the property we are after, as we discuss below (further discussion is found in the concluding section).

The first system we propose builds around Sangiorgi's system for uniform receptiveness [12]. However, we discard uniformity and introduce other constraints, as explained below. As expected, most difficulties in achieving responsiveness originate from responsive names being passed around. If an intended receiver of a responsive name r, say $a(x).P$, is not available "on time", r might never be delivered, hence used. In this respect, receptiveness is useful, because it can be used to ensure that inputs on a and on r are available as soon as they are created.

Even when delivery of r is ensured, however, one should take care that r will be processed properly. Indeed, the recipient might just "forget" about r, like in $(\nu a, r)(a(x).\mathbf{0} \mid \overline{a}\langle r\rangle)$; or r might be passed from one recipient to another, its use as a subject being delayed forever, like in

$$(\nu a, b, r)\big(\,!a(x).\overline{b}\langle x\rangle \mid !b(y).\overline{a}\langle y\rangle \mid \overline{a}\langle r\rangle\,\big)\,. \tag{1}$$

The first situation can be avoided by imposing that in the receiver $a(x).P$, name x occurs at least once in the body P. In fact, as we shall discuss in the paper, it is necessary that any responsive name be used *linearly*,

* The first autor is supported by the French government research grant ACI TRALALA. The second author is supported by the EU within the FET-GC2 initiative, project SENSORIA.

that is, it appears exactly once in input and once in output. Infinite delays like (1) can be avoided by using a stratification of names into *levels*, like in the type system for termination of Deng and Sangiorgi [4]. We will rule out divergent computations that involve responsive names infinitely often, but we'll do allow divergence in general.

Finally, even when a responsive name is eventually in place as subject of an output action, one has to make sure that such action becomes eventually available. In other words, one must avoid cyclic waiting like in

$$r(x).\bar{s}\langle x\rangle \mid s(y).\bar{r}\langle y\rangle\,. \tag{2}$$

This will be achieved by building a graph of the dependencies among responsive names and then checking for its acyclicity.

Receptiveness and linearity impose relevant limitations on the syntax of well-typed processes: nested free inputs are forbidden, as well as multiple outputs on the same name. On the other hand, the type system is expressive enough to enable a RPC programming style; in particular, we show that the usual CPS encoding of primitive recursive functions gives rise to well-typed processes.

In the second system we propose, the constraints on receptiveness and linearity are relaxed so as to allow certain forms of nested inputs and multiple outputs. For instance, the new system allows nondeterministic internal choice, which was forbidden in the first one. Relaxation of linearity and receptiveness raises new issues, though. As an example, responsiveness might fail due to "shortage" of inputs, like in (a, b and d responsive):

$$\bar{a}\langle b\rangle|\bar{a}\langle d\rangle|a(x).\bar{x}|b|d \xrightarrow{\tau}\xrightarrow{\tau} \bar{a}\langle d\rangle|d\,.$$

These issues must be dealt with by carefully "balancing" inputs and outputs in typing contexts and in processes. This system is flexible enough to encode into well-typed processes all orchestration patterns of Cook and Misra's ORC language [3]. Due to a rather crude use of levels, however, only certain forms of (tail-)recursion are encodable. In fact, neither the first system is subsumed by the second one, nor vice versa.

The rest of the paper is organized as follows. Syntax and operational semantics of the calculus are presented in Section 2, and the property we are after is formally defined. Section 3 introduces the first type system, after an informal discussion on the requirements for responsiveness. The main results, subject reduction and responsiveness, are presented in Section 4; there we give also a bound, depending on the size of a process, on the number of reductions necessary before a given responsive name is used. A simple extension of the first system (recursion on well-founded data) is presented in Section 5, where the encoding of primitive recursive functions is also discussed. The second system is presented in Section 6; several examples illustrating the extent and limits of the system are also discussed. We have no room for discussing the full encoding of ORC; hence only an example is shown. The concluding section contains some indications for further work, and a detailed discussion of related work.

2 Syntax and operational semantics

In this section we describe the syntax (processes and types) and the operational semantics of the calculus. On top of the operational semantics, we define the responsiveness property we are after.

2.1 Syntax

We focus on an asynchronous variant of the pi-calculus without nondeterministic choice. Indeed, asynchrony is a natural assumption in a distributed environment. Moreover, in the presence of a choice, it would be difficult to guarantee responsiveness of names that occur in branches that are discarded. A countable set of names $\mathcal{N}$, ranged over by $a,b,\ldots,x,y,\ldots$, is presupposed. The set $\mathcal{P}$ of *processes* $P,Q,\ldots$ is defined as the set of terms generated by the following grammar that obey the well-formedness conditions described below.

$$
\begin{array}{lll}
P ::= \ \mathbf{0} & & \textit{Inaction} \\
\quad \mid \overline{a}\langle b\rangle & & \textit{Output} \\
\quad \mid a(x).P & x \notin \mathrm{in}(P) & \textit{Input prefix} \\
\quad \mid\ !a(x).P & x \notin \mathrm{in}(P) & \textit{Replication} \\
\quad \mid P|Q & & \textit{Parallel} \\
\quad \mid (\nu b)P & & \textit{Restriction}
\end{array}
$$

In a non blocking output action $\overline{a}\langle b\rangle$, name a is said to occur in *output subject position* and b in *output object position*. In an input prefix $a(x).P$, and in a replicated input prefix $!a(x).P$, name a is said to occur in *input subject position* and x in *input object position*. We denote by $\mathrm{in}(P)$ the set of names occurring free in input subject position in P. The condition $x \notin \mathrm{in}(P)$, for input and replicated input, means that names can be passed around with the output capability only. This assumption simplifies reasoning on types and does not significantly affect the expressiveness of the language (see e.g. [2,10]). As usual, parallel composition, $P|Q$, represents the concurrent execution of P and Q and restriction, $(\nu b)P$, creates a fresh name b with initial scope P. Notions of free and bound names ($\mathrm{fn}(\cdot)$ and $\mathrm{bn}(\cdot)$), and α-equivalence ($=_\alpha$) arise as expected. In the paper, we shall only consider *well-formed* processes, where all bound names are distinct from each other and from free names. Please note that we do *not* identify processes up to α-equivalence (this means that an explicit operational rule that equates transition of α-equivalent processes will be needed).

Notationally, we shall often abbreviate $a(x).\mathbf{0}$ as $a(x)$, and $(\nu a_1)\ldots(\nu a_n)P$ as $(\nu a_1,\ldots,a_n)P$ or $(\nu \tilde{a})P$, where $\tilde{a} = a_1,\ldots,a_n$. In a few examples, the object part of an action may be omitted if not relevant for the discussion; e.g., $a(x).P$ may be shortened into $a.P$.

2.2 Sorts and types

The set of names $\mathcal{N}$ is partitioned into a family of countable *sorts* $\mathcal{S},\mathcal{S}',\ldots$. A fixed sorting à la Milner [11] is presupposed: that is, any sort $\mathcal{S}$ has an associated object sort $\mathcal{S}'$, and a name of sort $\mathcal{S}$ can only carry names of sort $\mathcal{S}'$. We only consider processes that are well-sorted in this system. Alpha-equivalence is assumed to be sort-respecting, in the obvious sense. Each sort is associated with a *type* T taken from the set $\mathcal{T}$ defined below. We write $a:\mathsf{T}$ if a belongs to a sort $\mathcal{S}$ with associated type T. The association between types and sorts is such that for each type there is at least one sort of that type.

A channel type $\mathsf{T}^{[\mathsf{u},k]}$ conveys three pieces of information: a type of carried objects T, a *usage* u, that can be *responsive* (ρ) or ω-*receptive* (ω), and an integer *level* $k \geq 0$. If $a:\mathsf{T}^{[\mathsf{u},k]}$ and $\mathsf{u} = \rho$ (resp. $\mathsf{u} = \omega$) we say that a is *responsive* (resp. ω-*receptive*). Informally, responsive names are guaranteed to be eventually used as subject in a communication, while ω-receptive names are guaranteed to be constantly ready to receive. Levels are used to bound the number of times a responsive name can be passed around, so to avoid infinite delay in their use as subject. We also consider a type I of *inert* names that cannot be used as subject of a communication – they just serve as tokens to be passed around. Finally, a type $\bot$ is introduced to collect those names that cannot be used at all: as we discuss below, $\bot$ is useful to formulate the subject reduction property while keeping the standard operational semantics.

Definition 1 (types). *The set $\mathcal{T}$ of* types *contains the constant $\bot$ and the set of terms generated by the grammar below. We use $T,S,\ldots$ to range over $\mathcal{T}$.*

$$
T ::= \quad I \quad \mid \quad T^U \qquad\qquad U ::= [\rho,k] \quad \mid \quad [\omega,k] \qquad (k \geq 0)
$$

2.3 Operational semantics

The semantics of processes is given by a labelled transition system in the early style, whose rules are presented in Table 1. An *action* μ can be of the following forms: free output, $\overline{a}\langle b\rangle$, bound output, $\overline{a}(b)$, input $a(b)$, or internal move τ. We define $\mathrm{n}(a(b)) = \mathrm{n}(\overline{a}\langle b\rangle) = \mathrm{n}(\overline{a}(b)) = \{a,b\}$ and $\mathrm{n}(\tau) = \emptyset$. A substitution σ is a finite

partial map from names to names; for any term P, we write $P\sigma$ for the result of applying σ to P, with the usual renaming convention to avoid captures.

The rules are standard, with a difference that we discuss in the following. The notation $\xrightarrow{\tau\langle a,b\rangle}$ is used to denote a τ-transition where the – free or bound – names a and b are used as subject and object, respectively, of a communication (we omit the formal definition of this notation, that can be given by keeping track of subject and object names in derivation of transitions.) In Rule (RES-ρ), a bound responsive subject a is alpha-renamed to a $\perp$-name c (a sort of "casting" of a to type $\perp$.) Informally, this alpha-renaming is necessary because in a well-typed process, due to the linearity constraint on responsive names, name a must vanish after being used as subject. The rule (RES) deals with the remaining cases of restriction. Note that if type and sorting information is ignored, one gets back the standard operational semantics of pi-calculus.

$$\text{(IN)}\ a(x).P \xrightarrow{a(b)} P[b/x] \qquad \text{(REP)}\ !a(x).P \xrightarrow{a(b)} !a(x).P|P[b/x]$$

$$\text{(OUT)}\ \overline{a}\langle b\rangle \xrightarrow{\overline{a}\langle b\rangle} \mathbf{0} \qquad \text{(ALPHA)}\ \frac{P =_\alpha Q \quad Q \xrightarrow{\mu} Q' \quad Q' =_\alpha P'}{P \xrightarrow{\mu} P'}$$

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

$$\text{(OPEN)}\ \frac{P \xrightarrow{\overline{a}\langle b\rangle} P' \quad a \neq b}{(\nu b)P \xrightarrow{\overline{a}(b)} P'} \qquad \text{(CLOSE}_1\text{)}\ \frac{P \xrightarrow{\overline{a}(b)} P' \quad Q \xrightarrow{a(b)} Q' \quad b \notin \mathrm{fn}(Q)}{P|Q \xrightarrow{\tau} (\nu b)(P'|Q')}$$

$$\text{(RES)}\ \frac{P \xrightarrow{\mu} P' \quad a \notin \mathrm{n}(\mu) \quad \mu = \tau\langle a,b\rangle \text{ implies } a \text{ not responsive}}{(\nu a)P \xrightarrow{\mu} (\nu a)P'} \qquad \text{(RES-}\rho\text{)}\ \frac{P \xrightarrow{\tau\langle a,b\rangle} P' \quad a \text{ responsive} \quad c:\perp \quad c \text{ fresh}}{(\nu a)P \xrightarrow{\tau\langle a,b\rangle} (\nu c)P'[c/a]}$$

Symmetric rules not shown.

Table 1. Rules for the labeled transition system

Notation We shall often refer to a silent move $P \xrightarrow{\tau} P'$ as a *reduction*. $P \xrightarrow{[a]} P'$ means $P \xrightarrow{\tau\langle a,b\rangle} P'$ for some free or bound name b. For a string $s = a_1 \cdots a_n \in \mathcal{N}^*$, $P \xrightarrow{[s]} P'$ means $P \xrightarrow{[a_1]} \cdots \xrightarrow{[a_n]} P'$, while $P \overset{[c]}{\Longrightarrow} P'$ means $P \xrightarrow{\tau}{}^* \xrightarrow{[c]} \xrightarrow{\tau}{}^* P'$. We use such abbreviations as $P \overset{[c]}{\Longrightarrow}$ to mean that there exists P' such that $P \overset{[c]}{\Longrightarrow} P'$.

We can now introduce the responsiveness property we are after. Informally, we think of a fair computation as a sequence of communications where for no name a a transition $\xrightarrow{[a]}$ is enabled infinitely often without ever taking place. Then a process uses a name in a responsive way if that name is eventually, that is, in all fair computations, used as subject of a communication. We then have the following definition. Below, we assume that any bound name occurring in P and s is distinct from any free name in P.

Definition 2 (responsiveness). *Let P be a process and $c \in \mathrm{fn}(P)$. We say that P* guarantees responsiveness of *c if whenever $P \xrightarrow{[s]} P'$ $(s \in \mathcal{N}^*)$ and c does not occur in s then $P' \overset{[c]}{\Longrightarrow}$.*

3 The type system $\vdash_1$

The type system consists of judgments of the form $\Gamma;\Delta \vdash_1 P$, where Γ and Δ are sets of names.

3.1 Overview of the system

Informally, names in Γ are those used by P in input, while in Δ are those used by P in output actions. There are several constraints on the usage of these names by P. A name in Γ must occur *immediately* (at top level) in input subject position, exactly once if it is responsive and replicated if it is ω-receptive. A responsive name in Δ must occur in P exactly once either in subject or in object output position, although not necessarily at top level, that is, occurrences in output actions underneath prefixes are allowed. There are no constraints on the use in output actions of ω-receptive names: they may be used an unbounded number of times, including zero. Linearity ("exactly once" usage) on responsive names is useful to avoid dealing with "dangling" responsive names, that might arise after a communication, like in (r responsive, object parts ignored):

$$(\nu r)(r.\mathbf{0}|\overline{r}|\overline{r}) \xrightarrow{\tau} (\nu r)(\mathbf{0}|\mathbf{0}|\overline{r})\,.$$

If the process on the LHS above were declared well-typed, this transition would violate the subject reduction property, as the process on the RHS above cannot be well-typed.

Linearity and receptiveness alone are not sufficient to guarantee a responsive usage of names. As discussed in the Introduction, we have also to avoid deadlock situations involving responsive names, like (2). This is simply achieved by building a *graph of dependencies* among responsive names of P (defined in the sequel) and checking for its acyclicity. We have also to avoid those situations described in the Introduction by which a responsive name is indefinitely "ping-pong"-ed among a group of replicated processes, like in (1). To this purpose, levels in types are introduced and the typing rules stipulate that sending a responsive name to a replicated input of level k may only trigger output of level less than k. This is similar to the use of levels in [4] to ensure termination. In our case, we just avoid divergent computations that involve responsive names infinitely often.

There is one more condition necessary for responsiveness, that is, the sets of input and output names must be "balanced", so as to ban situations like an output with no input counterpart. This constraint, however, is most easily formulated "on top" of well-typed-ness, and will be discussed later on.

3.2 Preliminary definitions

Formulation of the actual typing rules requires a few preliminary definitions. *Structural equivalence* is necessary in order to correctly formulate the absence of cyclic waiting on responsive names. We define structural equivalence $\equiv$ as the least equivalence relation satisfying the axioms below and closed under restriction and parallel composition. Let us point out a couple of differences from the standard notion [11]. First, there is no rule for replication ($!P \equiv P|!P$), as its right-hand side would not be well-typed. For a similar reason, in the rule $(\nu a)\mathbf{0} \equiv \mathbf{0}$ we require $a : \bot$ or $a : \mathsf{I}$.

$$\begin{array}{lll}
(\nu a)(P|Q) \equiv (\nu a)P|Q \quad \text{if } a \notin \mathrm{fn}(Q) & & (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \\
P|Q \equiv Q|P & & P|\mathbf{0} \equiv P \\
(P|Q)|R \equiv P|(Q|R) \quad P \equiv Q \quad \text{if } P =_\alpha Q & & (\nu a)\mathbf{0} \equiv \mathbf{0} \quad \text{if } a : \bot \text{ or } a : \mathsf{I}
\end{array}$$

Let us call a process P *prime* if either $P = \overline{a}\langle b\rangle$, or $P = a(x).P'$ or $P =!a(x).P'$. A process P is in *normal form* if $P = (\nu \tilde{d})(P_1|\cdots|P_n)$ ($n \geq 0$), every P_i is prime and $\tilde{d} \subseteq \mathrm{fn}(P_1,\ldots,P_n)$. Every process is easily seen to be structurally equivalent to a process in normal form.

In the dependency graph, defined below, nodes are responsive names of typing contexts and there is an arc from a to b exactly when an output action that involves a depends on an input action on b. Although the following definition does not mention processes, one should think of the pairs (Γ_i, Δ_i) below as typing contexts – limited to responsive names – for the P_i's in $P_1|\cdots|P_n$.

Definition 3 (dependency graph). *Let $\{(\Gamma_i, \Delta_i) : i = 1,\ldots,n\}$ be a set of context pairs. The* dependency graph $\mathrm{DG}(\Gamma_i, \Delta_i)_{i=1,\ldots,n}$ *is a graph (V, T) where: $V = \bigcup_{i=1,\ldots,n}(\Gamma_i \cup \Delta_i)$ is the set of nodes and $T = \bigcup_{i=1,\ldots,n}(\Gamma_i \times \Delta_i)$ is the set of arcs.*

We will have more to say on both structural equivalence and dependency graphs in Remark 1 at the end of the section. Like in [4], we will use a function $\mathrm{os}(P)$, defined below, that collects all – either free or bound – names in P that occur as subject of an *active* output action, that is, an output not underneath a replication (!).

$$\begin{aligned} \mathrm{os}(\mathbf{0}) &= \emptyset & \mathrm{os}(!a(b).P) &= \emptyset \\ \mathrm{os}(a(b).P) &= \mathrm{os}(P) & \mathrm{os}(\overline{a}\langle b\rangle) &= \{a\} \\ \mathrm{os}((\nu a)P) &= \mathrm{os}(P) & \mathrm{os}(P|Q) &= \mathrm{os}(P)\cup\mathrm{os}(Q) \end{aligned}$$

Finally, some notation for contexts and types. For any name a, we set $\mathrm{lev}(a) = k$ if $a : \mathsf{T}^{[\mathsf{u},k]}$ for some T and u, otherwise $\mathrm{lev}(a)$ is undefined. Given a set of names V, define $V^{\rho} \stackrel{\triangle}{=} \{\, x \in V \mid x \text{ is responsive} \,\}$ and $V^{\omega} \stackrel{\triangle}{=} \{\, x \in V \mid x \text{ is } \omega\text{-receptive} \,\}$. For V and W sets of names, we define $V \ominus W \stackrel{\triangle}{=} V \setminus W^{\rho}$. If $\Delta \cap \Delta' = \emptyset$, we abbreviate $\Delta \cup \Delta'$ as Δ, Δ' and if $a \notin \Delta$, we abbreviate $\Delta \cup \{a\}$ as Δ, a; similarly for Γ.

3.3 The typing rules

The type system is displayed in Table 2. Recall that each sort has an associated type. Linear usage of responsive names is ensured by rules (T-NIL) and (T-OUT), by the disjointness conditions in (T-PAR) and by forbidding responsive names to occur free underneath replication (T-REP). Absence of cyclic waiting involving responsive names is checked in (T-PAR). Finally, note the use of levels in rule (T-REP): communication involving a replicated input subject a and a responsive object can only trigger outputs of level less than $\mathrm{lev}(a)$. We say that a process P is *well-typed* if there are Γ and Δ such that $\Gamma;\Delta \vdash_1 P$ holds.

$$\text{(T-NIL)}\ \frac{\Gamma = \Delta^{\rho} = \emptyset}{\Gamma;\Delta \vdash_1 \mathbf{0}} \qquad \text{(T-OUT)}\ \frac{\Gamma = \emptyset \quad a,b \in \Delta \quad a : \mathsf{T}^{\mathsf{U}} \quad b : \mathsf{T} \quad \Delta^{\rho} - \{a,b\} = \emptyset}{\Gamma;\Delta \vdash_1 \overline{a}\langle b\rangle}$$

$$\text{(T-STR)}\ \frac{P \equiv Q \quad \Gamma;\Delta \vdash_1 Q}{\Gamma;\Delta \vdash_1 P} \qquad \text{(T-INP)}\ \frac{a : \mathsf{T}^{[\rho,k]} \quad b : \mathsf{T} \quad a \notin \Delta \quad \emptyset;\Delta,b \vdash_1 P}{a;\Delta \vdash_1 a(b).P}$$

$$\text{(T-RES-T)}\ \frac{a : \bot \quad \Gamma;\Delta \vdash_1 P}{\Gamma;\Delta \vdash_1 (\nu a)P} \qquad \text{(T-RES-I)}\ \frac{a : \mathsf{I} \quad \Gamma;\Delta,a \vdash_1 P}{\Gamma;\Delta \vdash_1 (\nu a)P} \qquad \text{(T-RES)}\ \frac{a : \mathsf{T}^{\mathsf{U}} \quad \Gamma,a;\Delta,a \vdash_1 P}{\Gamma;\Delta \vdash_1 (\nu a)P}$$

$$\text{(T-REP)}\ \frac{a : \mathsf{T}^{[\omega,k]} \quad b : \mathsf{T} \quad \Delta^{\rho} = \emptyset \quad \emptyset;\Delta,b \vdash_1 P \quad (\, b \text{ responsive implies } \forall c \in \mathrm{os}(P) : \mathrm{lev}(c) < k \,)}{a;\Delta \vdash_1 !a(b).P}$$

$$\text{(T-PAR)}\ \frac{\begin{array}{c} P = P_1|\cdots|P_n \ \ (n>1) \quad \forall i :\ P_i \text{ is prime and } \Gamma_i;\Delta_i \vdash_1 P_i \\ \forall i \neq j :\ \Gamma_i^{\rho} \cap \Gamma_j^{\rho} = \emptyset \text{ and } \Delta_i^{\rho} \cap \Delta_j^{\rho} = \emptyset \qquad \mathrm{DG}(\Gamma_i{}^{\rho},\Delta_i{}^{\rho})_{i=1,\ldots,n} \text{ is acyclic} \end{array}}{\bigcup_{i=1,\ldots,n} \Gamma_i \ ;\ \bigcup_{i=1,\ldots,n} \Delta_i \ \vdash_1 P}$$

Bound names in processes are assumed to be different from free names and from names in contexts.

Table 2. Typing rules of $\vdash_1$

Remark 1. (1) Avoiding deadlock on responsive names might be achieved by using levels in rule (T-INP), in the same fashion as in rule (T-REP), rather than using graphs. In fact, this would rule out cyclic waiting such as the one in (2) in the Introduction. We shall pursue this approach in the system of Section 6, where there is no way of defining a meaningful notion of dependency graph. However, in the present system this way of dealing with cyclic waiting would be unnecessarily restrictive, in particular it would ban as ill-typed the usual encoding of recursive functions into processes (see also Section 6.5).

(2) We note that, despite the presence of a rule for structural equivalence, the type system may be viewed as essentially syntax driven, in the following sense. Given P in normal form, $P = (\nu\tilde{d})(P_1|\cdots|P_n)$, and ignoring structural equalities that just rearrange the $\tilde{d}$ or the P_i's, there is at most one rule one can apply with P in the conclusion.

4 Subject reduction and responsiveness for system $\vdash_1$

Subject reduction states that well-typedness is preserved through reductions, and it is our first step towards proving responsiveness.

Theorem 1 (subject reduction). *Suppose* $\Gamma;\Delta \vdash_1 P$ *and* $P \xrightarrow{[a]} P'$. *Then* $\Gamma \ominus \{a\};\Delta \ominus \{a\} \vdash_1 P'$.

Our task is proving that any "balanced" well-typed process guarantees responsiveness (Definition 2) for all responsive names it contains.

Definition 4 (balanced processes). *A process* P *is* $(\Gamma;\Delta)$-balanced *if* $\Gamma;\Delta \vdash_1 P$, $\Gamma^\rho = \Delta^\rho$ *and* $\Delta^\omega \subseteq \Gamma^\omega$. *It is* balanced *if it is* $(\Gamma;\Delta)$*-balanced for some* Γ *and* Δ.

We need two main ingredients for the proof. The first one is given by the following proposition, stating that if the dependency graph of a process P is acyclic, then P always offers at least one output action involving a responsive name.

Proposition 1. *Suppose that* $\Gamma;\Delta \vdash_1 P$, *with* Γ, Δ *and* P *satisfying the conditions in the premise of rule* (T-PAR) *and* $\Gamma^\rho = \Delta^\rho$. *Then for some* $j \in \{1,\ldots,n\}$ *we have* $P_j = \overline{a}\langle b\rangle$ *with either* a *or* b *responsive.*

Next, we need a measure of processes that is decreased by reductions involving responsive names. We borrow from [4] the definition of *weight* of P, written $\mathrm{wt}(P)$: this is defined as a vector $\langle w_k, w_{k-1}, \ldots, w_0\rangle$, where $k \geq 0$ is the highest level of names in $\mathrm{os}(P)$, and w_i is the number of occurrences in output subject position of names of level i in P. A formal definition is given below. Here, "0_k" is an abbreviation for the vector $\langle 1,0,\ldots,0\rangle$ with k components "0" following "1" The vector with just one component that equals "0" is denoted by 0. Sum $+$ between two vectors is performed component-wise if they are of the same length; if not, the shorter one is first "padded" by inserting on the left as many 0's as needed.

$$\mathrm{wt}(\mathbf{0}) = 0 \qquad \mathrm{wt}(!a(b).P) = 0$$

$$\mathrm{wt}(a(b).P) = \mathrm{wt}(P) \qquad \mathrm{wt}(\overline{a}\langle b\rangle) = 0_k \text{ if } \mathrm{lev}(a) = k$$

$$\mathrm{wt}((\nu a)P) = \mathrm{wt}(P) \qquad \mathrm{wt}(P|Q) = \mathrm{wt}(P) + \mathrm{wt}(Q)$$

The set of all vectors can be ordered lexicographically. Assuming two vectors are of equal length (if not, the shorter vector is padded with 0's on the left), we define $\langle w_k,\ldots,w_0\rangle \prec \langle w'_k,\ldots,w'_0\rangle$ if there is i in $0,\ldots,k$ such that $w_j = w'_j$ for all $k \geq j > i$ and $w_i < w'_i$. This order is total and well-founded, that is, there are no infinite descending chains of vectors. The next proposition states that the weight of a process is decreased by reductions involving a responsive name, and leads us to Theorem 2, which is the main result of the section.

Proposition 2. *Suppose* $\Gamma;\Delta \vdash_1 P$ *and* $P \xrightarrow{\tau\langle a,b\rangle} P'$, *with either* a *or* b *responsive. Then* $\mathrm{wt}(P') \prec \mathrm{wt}(P)$.

Theorem 2 (responsiveness). *Let* P *be* $(\Gamma;\Delta)$*-balanced and* $r \in \Delta^\rho$. *Then* P *guarantees responsiveness of* r.

Proof. Assume $P \xrightarrow{[s']} Q$, for any Q, and $r \notin s'$. We have to show that $Q \xrightarrow{[r]}$. By contradiction, assume not. Let P' be a process with a *minimal* $\mathrm{wt}(\cdot)$ satisfying $Q \xrightarrow{[s'']} P'$ for some s'': this P' must exist by well-foundness of $\prec$. Moreover, $r \notin s''$. Let $s = s' \cdot s''$. By subject reduction we have that P' is $(\Gamma';\Delta')$-balanced, with $\Gamma' = \Gamma \ominus s$ and $\Delta' = \Delta \ominus s$.

Consider the normal form of the process P': $P' \equiv (\nu \tilde{d})(P_1|\cdots|P_n)$, where every P_i is prime and it must be $n > 1$, as r occurs in both input and output. By $\Gamma';\Delta' \vdash_1 P'$ we deduce $\Gamma', \tilde{d};\Delta', \tilde{d} \vdash_1 P_1|\cdots|P_n$ (by the typing rules for restriction and (T-STR)). By the typing rules, rule (T-PAR) must have been applied to infer the latter judgement, hence it must be: $(\Gamma', \tilde{d}) = \bigcup_{i=1,\ldots,n} \Gamma_i$, and $(\Delta', \tilde{d}) = \bigcup_{i=1,\ldots,n} \Delta_i$, and $\Gamma_i;\Delta_i \vdash_1 P_i$, where Δ_i^ρ (resp. Γ_i^ρ) are pairwise disjoint and $\mathrm{DG}(\Gamma_i{}^\rho, \Delta_i{}^\rho)_{i=1,\ldots,n}$ is acyclic. Moreover, $(\Delta', \tilde{d})^\rho = (\Gamma', \tilde{d})^\rho$ (by balancing), thus (Proposition 1) there is a j such that $P_j = \overline{a}\langle b\rangle$ with a or b responsive name. By $(\Delta', \tilde{d})^\omega \subseteq (\Gamma', \tilde{d})^\omega$ (again by balancing) and receptiveness of responsive and ω-receptive names ((T-INP), (T-REP), (T-OUT)), there is a k such that $P_k = (!)a(x).P'_k$. This implies $P' \xrightarrow{\tau\langle a,b\rangle} P''$, with $\mathrm{wt}(P'') \prec \mathrm{wt}(P')$, as either a or b is responsive (Proposition 2). But this is a contradiction, because P' was assumed to be a process with minimal weight satisfying $Q \xrightarrow{[s'']} P'$.

Next, we establish an upper bound on the number of steps that are always sufficient for a given responsive name to be used as subject. This upper bound can be given as a function of the syntactic size of P, written $|P|$, and of name levels in P. A similar result was given in [4] for terminating processes. Here, since we deal with processes that in general may not terminate, the upper bound must be given relatively to a notion of *scheduling* of transitions, that is introduced below.

Definition 5 (responsive scheduling). *A* responsive scheduling *is a finite or infinite sequence of reductions* $P_0 \xrightarrow{\tau\langle a_1,b_1\rangle} P_1 \xrightarrow{\tau\langle a_2,b_2\rangle} \cdots$ *where the bound names in* $\{(a_i,b_i)|i\geq 1\}$ *are all distinct from the free names in P and for each* $i\geq 0$*, either* a_i *or* b_i *is responsive.*

Theorem 3. *Let P be* $(\Gamma;\Delta)$*-balanced and* $r\in\Delta^\rho$ *and let k be the maximal level of names appearing in active output actions of P. Then there is at least one responsive scheduling that contains a reduction with r as subject. Moreover, in all such schedulings, the number of reductions preceding the reduction on r is upper-bounded by* $|P|^{k+1}$.

5 Recursion on well-founded data values

The system presented in Section 3 bans as ill-typed processes implementing recursive functions. As an example, consider the classical implementation of the factorial function, the process P below. For the purpose of illustration, we consider a polyadic version of the calculus enriched with `if...then...else`, natural numbers, variables $(x,y,...)$ and predicates/functions as expected. These extensions are straightforward to accommodate in the type system.

$$P \stackrel{\triangle}{=} !\,f(n,r).\texttt{if } n=0 \texttt{ then } \overline{r}\langle 1\rangle \texttt{ else } (\nu r')\big(\overline{f}\langle n-1,r'\rangle \,|\, r'(m).\overline{r}\langle m*n\rangle\big). \tag{3}$$

It would be natural to see f as ω-receptive and r and r' as responsive, but under these assumptions P would not be well-typed: the recursive call $\overline{f}\langle n-1,r'\rangle$ violates the constraint on levels of output actions under replication (rule (T-REP).) Nevertheless, it is natural to see an output $\overline{f}\langle n-1,r'\rangle$ triggered by a recursive call at f as "smaller" than the output $\overline{f}\langle n,r\rangle$ that has triggered it: at least, this is true if one takes into account the ordering relation on natural numbers. This means that the "weight" of the process decreases after each recursive call, and since natural numbers are well-founded, after some reductions no further recursive call will be possible, and a communication on r must take place. This idea from [4] is adapted here to our type system. For simplicity, we only consider the domain of natural values Nat. However, the results may be extended to any data type on which a well-founded ordering relation can be defined. We define an ordering relation "$<$" between (possibly open) integer expressions and variables as follows: $e<x$ if for each evaluation ρ under which e is defined, $e\rho<\rho(x)$. E.g., $x-1<x$. In the case of the monadic calculus, this relation is lifted to a "smaller than" relation $\lhd$ between output and input actions as follows. Below, d,d' denote either names or (open) expressions.

Definition 6 (ordering on actions). *We write* $\overline{c}\langle d\rangle \lhd a(d')$ *if either* $\mathrm{lev}(c)<\mathrm{lev}(a)$ *or* $\mathrm{lev}(c)=\mathrm{lev}(a)$ *and* $d=e<x=d'$.

The $\lhd$ relation is used in the typing rule below, that replaces rule (T-REP). We denote by $\mathrm{O}(P)$ the set of all output actions of P that are active, that is, not underneath a replication.

$$\text{(T-REP')}\quad \frac{\begin{array}{c} a:\mathsf{T}^{[\omega,k]} \quad b:\mathsf{T} \quad \Delta^\rho=\emptyset \quad \emptyset;\Delta,b\vdash_1 P \\ (b:\mathsf{Nat} \text{ or } b \text{ responsive}) \quad \text{implies} \quad \forall \overline{c}\langle d\rangle\in \mathrm{O}(P):\ \overline{c}\langle d\rangle \lhd a(b)\end{array}}{a;\Delta\vdash_1 !a(b).P}$$

In the polyadic case, $\lhd$ compares first the subject and then the object parts of two actions lexicographically (a different ordering is considered in [4].) As an example, it is easy to see that the process P in (3) is well-typed if $f:(\mathsf{Nat},\mathsf{Nat}^{[\rho,0]})^{[\omega,1]}$ and $r,r':\mathsf{Nat}^{[\rho,0]}$. The proof of responsiveness remains the same, modulo a change in function $\mathrm{wt}(\cdot)$ that takes into account contribution to weight given by the object part of active outputs. We omit the details of this definition.

Primitive Recursive Functions can be encoded into well-typed processes. The scheme of the encoding is an easy generalization of that seen above in (3) for the factorial function. More precisely, we have:

Proposition 3. *For every n-ary primitive recursive function f there is a well-typed process $\langle f \rangle_b$ such that: for each $(v_1,\ldots,v_n)$ in Nat^n the process $G \stackrel{\triangle}{=} (\nu b)(\langle f \rangle_b | \overline{b}\langle v_1,\ldots,v_n,r\rangle \mid r(x).\mathbf{0})$, with b ω-receptive and $r : (\mathsf{Nat})^{[\rho,k]}$ ($k \geq 0$), is balanced. Moreover, $f(v_1,\ldots,v_n) = m$ if and only if $G \xrightarrow{\tau}{}^{*} \xrightarrow{\overline{r}\langle m\rangle}$.*

6 Nested inputs, multiple outputs: the type system $\vdash_2$

The type system presented in Section 3 puts rather severe limitations on nesting of input actions and multiple use of names. These limitations stem from the "immediate receptiveness" and linearity conditions imposed on responsive names. For instance, the following encoding of internal choice $\overline{r}\langle a\rangle \oplus \overline{r}\langle b\rangle$, where r is responsive and a, b inert, is not well-typed

$$(\nu c)(\overline{c}\langle a\rangle | \overline{c}\langle b\rangle \mid c(x).\overline{r}\langle x\rangle)\,. \tag{4}$$

Limitations are also built-in in process syntax, as for example replicated outputs, that clearly violate linearity, are absent. These might be useful to encode situations like a process receiving from r a value y and storing it into a variable a, where reading from a means doing an input on a:

$$(\nu a)\big(r(x).!\overline{a}\langle x\rangle | a(y).P\big)\,. \tag{5}$$

For another example, a process that receives two values in a fixed order from two return channels, r_1 and r_2, and then outputs the max along s, may not be well-typed

$$r_1(x_1).r_2(x_2).\mathtt{if}\ x_1 \geq x_2\ \mathtt{then}\ \overline{s}\langle x_1\rangle\ \mathtt{else}\ \overline{s}\langle x_2\rangle\,. \tag{6}$$

We present below a new type system $\vdash_2$ that overcomes the limitations discussed above. In fact, we will trade off flexibility for expressiveness in terms of encodable functions, as only certain patterns of (tail-)recursion will be well-typed in the new system.

6.1 Syntax and operational semantics

We extend the syntax of processes by introducing replicated output and the syntax of types by introducing a new responsive usage of names, ρ^+, as follows:

$$P ::= \cdots \mid !\overline{a}\langle b\rangle$$
$$\mathsf{U} ::= \cdots \mid [\rho^+,k].$$

A name $a : \mathsf{T}^{[\rho^+,k]}$ is called *+-responsive*, as it is meant to be used *at least once* as subject of a communication. Therefore now we consider three different usages: ρ (for names used once), ρ^+ (for names used at least once) and ω (for names used an undefined number of times.) We point out that responsive names are not subsumed by +-responsive: in particular, as we shall see, the conditions on the type of carried objects are more liberal for responsive names. Operational semantics is enriched by adding the obvious rule for replicated output.

6.2 Overview of the system

We give here an informal overview of the system. In the type system, judgements are of the form $\Gamma;\Delta \vdash_2 P$ where in Γ and Δ each +-responsive name a is annotated with a *capability* t, written a^t. A capability t can be one of four kinds: n (*null*), s (*simple*), m (*multiple*), p (*persistent*). Informally, capabilities have the following meaning (in the examples below, we ignore object parts of some actions and assume b is a (+-)responsive name):

- a^{n} indicates that a cannot be used at all. This capability has been introduced to uniformly account for +-responsive names that disappear after being used as subjects.
- a^{s} indicates that a appears at least once, but never under a replication. Examples: $a.P$, $b.a.P$, $\overline{a}$ and $b.\overline{a}$.
- a^{m} indicates that a appears at least once, even under replication, but never as a subject of a replicated action. Examples: $a.P|a.Q$, $!b.a.P$ and $!b.\overline{a}$.
- a^{p} indicates that a only appears as a subject of a replicated action. Examples: $!a.P$, $!\overline{a}$, $b.!\overline{a}$ and $!b.!\overline{a}$.

Note that a name a may be given distinct capabilities in input (Γ) and output (Δ). E.g. one may have, again ignoring the object parts, $\Gamma;\Delta \vdash_2 !a.P|\overline{a}$, where $a^{\mathsf{p}} \in \Gamma$ and $a^{\mathsf{s}} \in \Delta$. Next we illustrate and motivate the constraints on names usage realized by the typing rules and by the balancing conditions discussed later on. There are essentially three of them:

1. If $a^{\mathsf{m}} \in \Gamma$ then $a^{\mathsf{p}} \in \Delta$. This is to avoid deadlocks arising from not having enough output actions of subject a, like in (a and b +-responsive names):
$$a\,|\,a.b\,|\,\overline{a}\,|\,\overline{b} \xrightarrow{\tau} a.b\,|\,\overline{b} \not\xrightarrow{\tau} .$$
This situation is in fact avoided if a appears in replicated output subject, like in $a\,|\,a.b\,|\,!\overline{a}\,|\,\overline{b}$.
2. If $a^t \in \Gamma$ and a carries (+-)responsive names, then $t = \mathsf{p}$. This is to avoid deadlocks arising from having too many outputs of subject a that carry (+-)responsive names, like in (a +-responsive, b and d (+-)responsive names):
$$\overline{a}\langle b\rangle|\overline{a}\langle d\rangle|a(x).\overline{x}|b|d \xrightarrow{\tau}\xrightarrow{\tau} \overline{a}\langle d\rangle|d\,.$$
3. Names occurring under an (either simple or replicated) input must be of smaller level than the input subject. The role of this condition is twofold, now. Under replicated inputs, it avoids infinite delays, like in the first system. Under simple inputs, it serves to avoid cyclic waiting, like in (a, b (+-)responsive): $a.\overline{b}|b.\overline{a}$. This was achieved by the use of dependency graphs in the first system. As announced in Remark 1, however, there appear to be no meaningful extension of this notion of graph in the present system. In particular, acyclicity of the graph might not be preserved by reductions. E.g. consider the reduction on c of the process below:
$$b(x).\overline{a}\langle x\rangle|c(x).a(y).\overline{x}\langle y\rangle|\overline{c}\langle b\rangle\,.$$

There are other constraints that have been introduced for technical convenience (essentially, to avoid divergences and deadlocks difficult to control in the proof of responsiveness) and that shall not be discussed further:

(a) names with input capability s (simple) occur exactly once in input subject position;
(b) names with capability p (persistent) occur exactly once in subject position, either in input or in output, but not in both; this also implies that persistent names cannot be passed around. Moreover, free replicated inputs cannot be guarded.

The above conditions are often met in applications (e.g., they are in the encoding of ORC presented in the next section.)

6.3 The typing rules

We first introduce some additional notations. Contexts Γ and Δ are sets of annotated names of the form a^t, where t is a capability. Each name occurs at most once in a context. +-responsive names are annotated with one of the four capabilities n, s, m or p, while non-+-responsive names are always annotated with a default "$-$" capability; when convenient a^- is abbreviated simply as a. Union and intersection of two contexts, written $\Gamma_1 \cup \Gamma_2$ and $\Gamma_1 \cap \Gamma_2$, are defined only if the contexts agree on capabilities of common names, that is whenever $a^{t_i} \in \Gamma_i$ for $i = 1,2$ then $t_1 = t_2$. We write Γ_1, Γ_2 in place of $\Gamma_1 \cup \Gamma_2$ if $\Gamma_1 \cap \Gamma_2 = \emptyset$; while Γ_1, a^t abbreviates $\Gamma_1, \{a^t\}$. For any context Γ and capability t, we define $\Gamma^t \triangleq \{a | a^t \in \Gamma\}$. The set of names $\Gamma^{\mathsf{p}^+} \triangleq \{\, a \mid a$ is +-responsive and $a^t \in \Gamma$ for some $t \neq \mathsf{n}\,\}$ and Γ^{p}, Γ^{ω} (defined similarly) will also be useful. The typing rules are presented in Table 3.

6.4 Subject reduction and responsiveness

Subject reduction carries over to the new system, modulo a small notational change. For Γ a typing context and V a set of names let us denote by $\Gamma \ominus^+ V$ the typing context obtained by removing from Γ each a^t such that $a \in V$. Let us denote by $\mathrm{on}(P)$ the set of names occurring free in output position in P.

Theorem 4 (subject reduction for system $\vdash_2$). *$\Gamma;\Delta \vdash_2 P$ and $P \xrightarrow{[a]} P'$ imply $\Gamma';\Delta' \vdash_2 P'$, with $\Gamma' = \Gamma \ominus^+ (\{a\} \setminus \mathrm{in}(P'))$ and $\Delta' = \Delta \ominus^+ (\{a\} \setminus \mathrm{on}(P'))$.*

$$(\mathrm{T}_+\text{-INP})\ \frac{a:\mathsf{T}^{[\mathsf{u},k]}\ \text{with}\ \mathsf{u}\neq\omega \quad b:\mathsf{T} \quad \forall c\in \mathrm{os}(P)\cup \mathrm{in}(P):\ \mathrm{lev}(c)<k \qquad \Gamma^{\mathsf{p}}=\Gamma^{\omega}=\emptyset \quad a\ \text{+-responsive implies}\ b\ \text{not (+-)responsive} \qquad \Gamma;\Delta,b^{t'}\vdash_2 P \quad t\neq \mathsf{n},\mathsf{p} \quad t'\neq \mathsf{n},\mathsf{p}}{\Gamma,a^{t};\Delta\vdash_2 a(b).P}$$

$$(\mathrm{T}_+\text{-REP})\ \frac{a:\mathsf{T}^{[\omega,k]} \quad b:\mathsf{T} \quad \Delta^{\mathsf{p}}=\Delta^{\mathsf{p}^+}=\emptyset \quad \emptyset;\Delta,b^{t'}\vdash_2 P \quad t'\neq \mathsf{n},\mathsf{p} \qquad b\ \text{(+-)responsive implies}\ \forall c\in \mathrm{os}(P):\ \mathrm{lev}(c)<k}{a^{-};\Delta\vdash_2 !a(b).P}$$

$$(\mathrm{T}_+\text{-REP}^{\mathsf{p}})\ \frac{a:\mathsf{T}^{[\mathsf{p}^+,k]} \quad b:\mathsf{T} \quad \Gamma^{\ell}=\emptyset\ \text{for}\ \ell\in\{\mathsf{p},\omega,\mathsf{s},\mathsf{p}\} \quad \Delta^{\ell'}=\emptyset\ \text{for}\ \ell'\in\{\mathsf{s},\mathsf{p},\mathsf{p}\} \qquad \Gamma;\Delta,b^{t}\vdash_2 P \quad t\neq \mathsf{n},\mathsf{p} \quad \forall c\in \mathrm{os}(P)\cup \mathrm{in}(P):\ \mathrm{lev}(c)<k}{\Gamma,a^{\mathsf{p}};\Delta\vdash_2 !a(b).P}$$

$$(\mathrm{T}_+\text{-OUT})\ \frac{a:\mathsf{T}^{\mathsf{U}} \quad b:\mathsf{T} \quad \Delta^{\mathsf{p}}=\Delta^{\mathsf{p}^+}=\emptyset \quad t'\neq \mathsf{n},\mathsf{p} \quad t\neq \mathsf{n},\mathsf{p}}{\emptyset;\Delta,a^{t},b^{t'}\vdash_2 \overline{a}\langle b\rangle}$$

$$(\mathrm{T}_+\text{-NIL})\ \frac{\Delta^{\mathsf{p}}=\Delta^{\mathsf{p}^+}=\emptyset}{\emptyset;\Delta\vdash_2 \mathbf{0}} \qquad (\mathrm{T}_+\text{-OUT}^{\mathsf{p}})\ \frac{a:\mathsf{T}^{[\mathsf{p}^+,k]} \quad b:\mathsf{T} \quad \Delta^{\mathsf{p}}=\Delta^{\mathsf{p}^+}=\emptyset \qquad b\ \text{not (+-)responsive}}{\emptyset;\Delta,a^{\mathsf{p}},b^{-}\vdash_2 !\overline{a}\langle b\rangle}$$

$$(\mathrm{T}_+\text{-RES-}\bot)\ \frac{a:\bot \quad \Gamma;\Delta\vdash_2 P}{\Gamma;\Delta\vdash_2 (\nu a)P} \qquad (\mathrm{T}_+\text{-RES})\ \frac{a:\mathsf{T}^{\mathsf{U}} \quad \Gamma,a^{t};\Delta,a^{t'}\vdash_2 P}{\Gamma;\Delta\vdash_2 (\nu a)P}$$

$$(\mathrm{T}_+\text{-RES-I})\ \frac{a:\mathsf{I} \quad \Gamma;\Delta,a^{-}\vdash_2 P}{\Gamma;\Delta\vdash_2 (\nu a)P} \qquad (\mathrm{T}_+\text{-WEAK-}\Gamma)\ \frac{\Gamma;\Delta\vdash_2 P}{\Gamma,a^{\mathsf{n}};\Delta\vdash_2 P} \qquad (\mathrm{T}_+\text{-WEAK-}\Delta)\ \frac{\Gamma;\Delta\vdash_2 P}{\Gamma;\Delta,a^{\mathsf{n}}\vdash_2 P}$$

$$(\mathrm{T}_+\text{-PAR})\ \frac{\Gamma=\Gamma_1\cup\Gamma_2 \quad \Delta=\Delta_1\cup\Delta_2 \quad \Gamma_i;\Delta_i\vdash_2 P_i \quad (i=1,2) \qquad \Gamma_1^{\ell}\cap\Gamma_2^{\ell}=\emptyset\ \text{for}\ \ell\in\{\mathsf{p},\mathsf{s},\mathsf{p}\} \qquad \Delta_1^{\ell'}\cap\Delta_2^{\ell'}=\emptyset\ \text{for}\ \ell'\in\{\mathsf{p},\mathsf{p}\} \qquad \Gamma^{\mathsf{p}}\cap\Delta^{\mathsf{p}}=\emptyset \quad \Gamma^{\mathsf{m}}\cap(\Delta^{\mathsf{s}}\cup\Delta^{\mathsf{m}})=\emptyset}{\Gamma;\Delta\ \vdash_2\ P_1|P_2}$$

Bound names in processes are assumed to be different from free names and from names in contexts.

Table 3. Typing rules of $\vdash_2$

The balancing requirements are now more stringent. They include those for responsive and ω-receptive names necessary in the first system (condition 1 below). Concerning +-responsive names, "perfect balancing" between input and output is required only for those names that carry (+-)responsive names (condition 2). Moreover, the same requirements apply also to restricted +-responsive names (condition 3).

Given a set of names V let us define $V^{\dagger}=\{a\in V\,|\,a:\mathsf{T}$ and T is of the form $(\mathsf{S}^{[\mathsf{u},k]})^{[\mathsf{u}',h]}$ with $\mathsf{u}\in\{\mathsf{p},\mathsf{p}^+\}\ \}$. Define $\mathrm{r}_{\mathrm{i}}^{+}(P)$ (resp. $\mathrm{r}_{\mathrm{o}}^{+}(P)$) as the set of restricted +-responsive names in P occurring in an input (resp. output) action in P, even underneath a replication. We have the following definition and result.

Definition 7 (strongly balanced processes). *A process P is* $(\Gamma;\Delta)$-strongly balanced *if $\Gamma;\Delta\vdash_2 P$ and the following conditions hold: (1) $\Gamma^{\mathsf{p}}=\Delta^{\mathsf{p}}$ and $\Delta^{\omega}\subseteq\Gamma^{\omega}$; (2) $\Gamma^{\mathsf{p}^+}\subseteq\Delta^{\mathsf{p}^+}$ and $(\Delta^{\mathsf{p}^+})^{\dagger}\subseteq(\Gamma^{\mathsf{p}^+})^{\dagger}$; (3) $\mathrm{r}_{\mathrm{i}}^{+}(P)\subseteq\mathrm{r}_{\mathrm{o}}^{+}(P)$ and $(\mathrm{r}_{\mathrm{o}}^{+}(P))^{\dagger}\subseteq(\mathrm{r}_{\mathrm{i}}^{+}(P))^{\dagger}$.*

The proof of the following theorem is non-trivial, as strong balancing is preserved through reductions only up to certain transformations on processes.

Theorem 5 (responsiveness for system $\vdash_2$). *Suppose P is $(\Gamma;\Delta)$-strongly balanced and $r \in \Delta^{\rho} \cup \Gamma^{\rho^+}$. Then P guarantees responsiveness of r.*

6.5 Examples

Let us now examine a few examples. In what follows, unless otherwise stated we assume that x,y are of type inert, that a,b,c are +-responsive and that r,s are responsive. Conditions on levels are ignored when obvious. Process (4) at the beginning of the section is well-typed with c of capability multiple in output and simple in input; it is strongly balanced if put in parallel with an appropriate context of the form $r(x).P$. Process (5) is well-typed with a of capability persistent in output and simple in input (also, P must be assumed strongly balanced, and not containing free persistent inputs or names of level greater than a's); it is strongly balanced if put in parallel with $\overline{r}\langle x\rangle$. Process (6) is well-typed assuming r_1 and r_2 of capability simple in input and x_1,x_2 natural number variables (the obvious extension of the system with `if-then-else` and naturals is here assumed); again, it is strongly balanced if put in parallel with an appropriate context.

The next two examples involve non-linear usages of +-responsive names arising from replication and reference passing. We mention these examples also because they will help us to compare our system to existing type systems that enforce lock freedom, a property related to responsiveness (see the concluding section). The first example involves only replication, object parts play no role:

$$!a.\overline{b} \,|\, \overline{a} \,|\, b\,. \tag{7}$$

The above process is strongly balanced under the assumption that a has capability persistent in input and simple/multiple in output, and b has capability simple in input and multiple in output; also, the level of b must be less than a's. In the next example, an agent sort of "looks up" a directory a to get the address of a service b, and then calls this service:

$$!a(z).\overline{z}\langle b\rangle \,|\, (\nu r)(\overline{a}\langle r\rangle \,|\, r(w).\overline{w}) \,|\, b\,. \tag{8}$$

This process is strongly balanced under the assumption that: a is persistent in input and simple or multiple in output; b is simple in input and multiple in output; also, the level of b must be smaller than $r's$ and the level of r must be smaller than a's (the variant where b is replaced by $!b$ is also strongly balanced; in this case b is persistent in input.)

The type system $\vdash_2$ can be extended to the polyadic version of the calculus with naturals and variables exactly as seen in Section 5, i.e. by using the "$\lhd$" relation over actions in rules (T_+-INP), (T_+-REP) and (T_+-REP$^{\mathsf{P}}$). Now, consider the factorial function in (3) and assume r,r' are (+-)responsive. It is easily seen that (3) is not well-typed in the present system: in fact, because of the recursive call at f, it cannot be $\mathrm{lev}(r) < \mathrm{lev}(r')$. In general, the type system bans as ill-typed recursive calls of the form $g(h(g(i),i))$, thus ruling out the usual encoding of primitive recursion. Certain forms of recursion, like the tail-recursive version of factorial below, are however still well-typed

$$!f(x,a,r).\texttt{if } x=0 \texttt{ then } \overline{r}\langle a\rangle \texttt{ else } \overline{f}\langle x-1, a*x, r\rangle\,.$$

7 Encoding the Structured Orchestration Language

In this section we show that the orchestration patterns definable in the ORC language [3] can be encoded into π-calculus processes that are well-typed in system $\vdash_2$. For the sake of simplicity, we suppose that inert names, ranged over by c, are the only data values that can be exchanged among ORC services. ORC terms, ranged over by $f,g,\ldots$, are defined by the following grammar, where x ranges over variables, and p over variables and names ($p ::= x|c$):

$f,g ::= \mathbf{0} \,\big|\, M(p) \,\big|\, E(p)$		inaction, site and expression calls
	$\big|\, \mathrm{let}(p)$	publication
	$\big|\, f > x > g$	sequential composition
	$\big|\, f \,|\, g$	symmetric parallel composition
	$\big|\, g \text{ where } x :\in f$	asymmetric parallel composition

$$[\![\mathrm{let}(x)]\!]_s = x(y).\bar{s}\langle y\rangle \qquad [\![\mathrm{let}(c)]\!]_s = \bar{s}\langle c\rangle$$

$$[\![E(x)]\!]_s = x(y).\overline{E}\langle y,s\rangle \qquad [\![E(c)]\!]_s = \overline{E}\langle c,s\rangle$$

$$[\![M(x)]\!]_s = x(y).[\![M(y)]\!]_s \qquad [\![M(c)]\!]_s = (\nu r)(\overline{M}\langle c,r\rangle \,|\, r(y).\bar{s}\langle y\rangle)$$

$$[\![f\,|\,g]\!]_s = [\![f]\!]_s \,|\, [\![g]\!]_s \qquad [\![g \,\mathrm{where}\, x :\in f]\!]_s = (\nu r)\big([\![f]\!]_r \,|\, (\nu x)(r(y).!\bar{x}\langle y\rangle \,|\, [\![g]\!]_s)\big)$$

$$[\![f > x > g]\!]_s = (\nu t)\big([\![f]\!]_t \,|\, !t(y).(\nu x)(!\bar{x}\langle y\rangle \,|\, [\![g]\!]_s)\big)$$

Name	c,y	x	s	r	t	E	M
Type	I	$\mathsf{I}^{[\rho^+,k_x]}$	$\mathsf{I}^{[\rho^+,h]}$	$\mathsf{I}^{[\rho^+,h']}$	$\mathsf{I}^{[\rho^+,h']}$	$(\mathsf{I},\mathsf{I}^{[\rho^+,h]})^{[\rho^+,k_E]}$	$(\mathsf{I},\mathsf{I}^{[\rho^+,h']})^{[\rho^+,k_M]}$
Input Cap.		m	s	s	p	p	p
Output Cap.	$-$	p	$\mathsf{s/m}$	$\mathsf{s/m}$	$\mathsf{s/m}$	$\mathsf{s/m}$	$\mathsf{s/m}$

with: $k_x > h, k_E, k_M$, and $k_E > h$, and $k_M > h, h'$ and $h' > h, k_x$

Table 4. Encoding of the ORC language and typing assumptions.

Here M is a *site* name, p is a parameter (variable or name) and for every expression name E there exists a declaration $E(x) \triangleq f$, where x is the formal parameter. The language's primitives can be informally explained as follows (for a formal definition of ORC's operational semantics, the reader is referred to [3].) Each closed expression *publishes* (returns) a sequence of zero or more values. A site call $M(p)$ publishes a predefined value associated with site M. An expression call $E(p)$ publishes the values retuned by $f[p/x]$. The expression $\mathrm{let}(c)$ publishes the value c. In $f > x > g$, the execution of f is started, and every value c published by f triggers a new instance of g, $g[c/x]$; the sequence of values produced by all these instances running in parallel is published. In $f|g$, the sequence obtained by interleaving values produced by f and g is published. In g where $x :\in f$ the values produced by g are published; however, the execution of f and g is started in parallel, and each subterm of g that depends on x is blocked until f produces the first value v, which causes x to be replaced by v; subsequent values published by f are discarded.

ORC terms are translated into π-calculus by the function $[\![\cdot]\!]_s$ defined in Table 4; s is used here as a result channel. Encoding of a declaration $E(x) \triangleq f$ is given by $!E(x,s).[\![f]\!]_s$. For simplicity, we assume that a site M receives something and then publishes a predefined value c and returns, thus the encoding of the site M is simply $!M(x,s).\bar{s}\langle c\rangle$. The encoded terms are well-typed under typing assumption in Table 4. Levels are left unspecified, but suitable values for them can be easily inferred by inspection. Capability $\mathsf{s/m}$ indicates the possibility of having either s or m, depending on whether the name occurs under a replication (m) or not ($\mathsf{s/m}$). E.g. capabilities associated to free names appearing in g in the encoding of $f > x > g$ must be given capability m. The following result can be used for reasoning about responsiveness of ORC expressions. In what follows, given an ORC term f, D_f stands for the parallel composition of the encodings of all declarations and sites involved in the definition of f and $\tilde{d} = \mathrm{fn}(D_f)$. We write $f \stackrel{!c}{\Rightarrow}$ if a term f publishes the value c possibly after some internal reductions.

Proposition 4. *Let f be a closed* ORC *term and suppose D_f is well-typed. Under the typing assumptions of Table 4, $[\![f]\!]_s$ is well-typed and $F \stackrel{\triangle}{=} (\nu\tilde{d})([\![f]\!]_s \,|\, D_f \,|\, s(x).\mathbf{0})$, with s and $\tilde{d}$ +-responsive, is strongly balanced. Moreover, $f \stackrel{!c}{\Rightarrow}$ if and only if $F \stackrel{\bar{s}\langle c\rangle}{\Longrightarrow}$.*

In the following we show an example of encoding of an orchestration pattern defined in the ORC language [3] into a π-calculus process that is well-typed in system $\vdash_2$. Consider the ORC function below, which emails n times the first newspage from *CNN* or *BBC* to address a and publish the current value of n after every sending and at the end of the cicle (we leave aside the specifications of sites *BBC*, *CNN*, ...):

$$\mathit{MailNews}(n,a) \triangleq \text{ if } n = 0 \text{ then } \mathrm{let}(n)$$
$$\text{else } \mathit{MailNews}(n-1,a) \,|\, \mathit{Mail}(t,a) >> \mathrm{let}(n) \text{ where } t :\in (\mathit{CNN}|\mathit{BBC}) \,.$$

Suppose the encodings of sites *CNN*, *BBC* and *Mail* are respectively $!CNN(x).\bar{x}\langle N\rangle$, $!BBC(x).\bar{x}\langle N'\rangle$ and $!Mail(x,a,r).[send\ x\ to\ a].\bar{r}$, where N and N' represent news. The function *MailNews* can be encoded as follows:

$$MN \stackrel{\triangle}{=} !Mn(n,a,s).\text{if } n=0 \text{ then } \bar{s}\langle n\rangle \text{ else } \Big(\overline{Mn}\langle n-1,a,s\rangle \mid (\nu r)\Big(\overline{CNN}\langle r\rangle \mid \overline{BBC}\langle r\rangle \mid (\nu t)\big(r(y).!\bar{t}\langle y\rangle \mid (\nu r')(t(x).\overline{Mail}\langle x,a,r'\rangle \mid !r'(x).\bar{s}\langle n\rangle)\big)\Big)\Big) .$$

where the received channel s is used for publishing values. As in the previous example, consider the extension of type system $\vdash_2$ with values and polyadic communication; *MN* is well-typed supposing s, r, r' and t $+$-responsive, $\text{lev}(Mn) > \text{lev}(CNN)$, $\text{lev}(Mn) > \text{lev}(BBC)$ and $\text{lev}(CNN), \text{lev}(BBC) > \text{lev}(r) > \text{lev}(t) > \text{lev}(Mail) > \text{lev}(r') > \text{lev}(s)$.

8 Conclusions and related works

We have presented two type systems for statically enforcing responsive usage of names in pi-calculus processes. The first system combines linearity, receptiveness and techniques for deadlock and livelock avoidance. In the second system, receptiveness and linearity are relaxed at the price of stronger requirements on levels and balancing: we lose some expressive power in terms of encodable recursive functions, but are able to type interesting processes, such as translation of ORC terms. Both systems are syntax driven, so that type checking should be straightforward and efficient to implement. Extensions with type inference and subtyping deserve further investigation, mainly due to the presence of levels.

Beside the works, already discussed, on receptiveness [12] and termination [4], there are a few more works related to ours and that are discussed below.

Closely related to our system one are a series of papers by Berger, Honda and Yoshida on linearity-based type systems. In [16], they introduce a type system that guarantees termination and determinacy of pi-calculus processes, i.e. *Strong Normalization* (SN). Our techniques of system one are actually close to theirs, as far as the linearity conditions and cycle-detection graphs are concerned (see also the type system in [14]). However SN is stronger than responsiveness, in particular SN implies responsiveness on all linear names under a balancing condition. In fact, the system in [16] is stricter than our system one, e.g. it does not allow linear subjects to carry linear objects, and bans ω-names, hence any form of nondeterminism and divergence, as these features would obviously violate SN. Yoshida's type system in [15], in turn a refinement of the systems in [16] and [1], is meant to ensure a *Linear Liveness* property, meaning that the considered process eventually prompts for a free output at a given channel. This property is related to responsiveness, the difference being that Linear Liveness does not imply synchronization, hence the corresponding input might not become available. Two kinds of names are considered in [15]: linear (used exactly once) and *affine* (used at most once). Linear subjects carrying linear objects are forbidden and internal mobility is assumed – only restricted names can be passed around.

Closely related to our system two are a series of papers by Kobayashi and collaborators. A type system for linearity in the pi-calculus was first introduced in [9]. This system can be used to ensure that any linear name in a process occurs exactly once in input and once in output; however, it cannot ensure that a linear name will be eventually used as a subject of a synchronization. Kobayashi's type systems in [5,6] can be used to guarantee that, under suitable fairness assumptions, certain actions are lock free, i.e. are deemed to succeed in synchronization, if they become available ([7] is a further refinement, but the resulting system cannot be used to enforce responsiveness.) Channel types are defined in terms of *usages*: roughly, CCS-like expressions on the alphabet $\{I,O\}$, that define the order in which each channel must be used in input (I) and in output (O). Each I/O action is annotated with an *obligation* level, related to when the action must become available, and a *capability* level, related to when the action must succeed in synchronization if it becomes available. A level can be a natural number or infinity, the latter used to annotate actions that are not guaranteed to become available/succeed in synchronization. This scheme is fairly general, allowing e.g. for typing of shared-memory structures such as locks and semaphores, which are outside the scope of our systems. Concerning responsiveness, on the other hand, it appears that our $+$-responsive types cannot in general be encoded into lock-freedom types. More precisely, one can exhibit processes well-typed in our system two and containing $+$-responsive names that cannot be assigned

a finite capability in Kobayashi's systems. For example, both the process (7) and the "service-lookup" (8) are well-typed (in fact, strongly balanced) in our system two, under a typing context where b is $+$-responsive. They are not in the systems of [5,6], under any type context that assigns to b a finite capability: the reason is that a finite-capability input on b is required to be balanced by an instance of a finite-obligation output $\overline{b}$, that cannot be statically determined in the given processes (although, after the submission of the present paper, the `TyPiCal` tool [8] has been modified to handle also processes of this form.) Another difference from [5,6] is that these systems partly rely on a form of dynamic analysis which is performed on types: the *reliability* condition on usages, which roughly plays the same role played in our systems by balancing, is checked via a reduction to the reachability problem for Petri Nets. As previously remarked, our systems are entirely static.

Acknowledgments We wish to thank Davide Sangiorgi and Naoki Kobayashi for stimulating discussions on the topics of the paper.

References

1. M. Berger, K. Honda and N. Yoshida. Sequentiality and the π-calculus. In *Proc. of TCLA*, 2001. *LNCS*, 2044, pp.29-45.
2. M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science*, 195(2):205-226, 1998.
3. W. R. Cook and J. Misra. Computation Orchestration: A Basis for Wide-Area Computing. *Journal of Software and Systems Modeling*, 2006. `http://www.cs.utexas.edu/~wcook/projects/orc/`.
4. Y. Deng and D. Sangiorgi. Ensuring Termination by Typability. In *Proc. of IFIP TCS*, pp.619-632, 2004. Full version in *Information and Computation*, 204(7):1045-1082, 2006.
5. N. Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122-159, 2002.
6. N. Kobayashi. Type-Based Information Flow Analysis for the Pi-Calculus. *Acta Informartica*, 42(4-5): 291-347, 2005.
7. N. Kobayashi. A New Type System for deadlock-Free Processes. To appear in *Proc. of CONCUR*, 2006.
8. N. Kobayashi. The `TyPiCal` tool, available at `http://www.kb.ecei.tohoku.ac.jp/~koba/typical/`.
9. N. Kobayashi, B.C. Pierce and D.N. Turner. Linearity and the Pi-Calculus. In *Proc. of POPL*, 1996. Full version in *ACM Transactions on Programming Languages and Systems*, 21(5):914-947, 1999.
10. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proc. of ICALP*, 1998. *LNCS*, 1443, pp.856-867. Full version in *Mathematical Structures in Computer Science*, 14(5):715-767, 2004.
11. R. Milner. The polyadic π-calculus: a tutorial. Tec.Rep. LFCS report ECS-LFCS-91-180, 1991. Also in *Logic and Algebra of Specification*, Springer-Verlag, pp.203-246, 1993
12. D. Sangiorgi. The name discipline of uniform receptiveness. In *Proc. of ICALP*, 1997. *TCS*, 221(1-2):457-493, 1999.
13. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.
14. N. Yoshida. Graph Types for Monadic Mobile Processes. In *Proc. of 16th FST/TCS*, *LNCS*, 1180, pp.371-386, 1996.
15. N. Yoshida. Type-Based Liveness in the Presence of Nontermination and Nondeterminism. MCS Technical Report, 2002-20, University of Leicester, 2002.
16. N. Yoshida, M. Berger and K. Honda. Strong Normalisation in the π-calculus. In *Proc. of LICS*, 2001. *IEEE*, pp.311-322.

A Proof of Theorem 5

We need two preliminary propositions, whose proofs are omitted. The first proposition ensures that processes strongly balanced under nontrivial contexts always have a reduction involving a (+-)responsive name.

Proposition A1 *Suppose P is $(\Gamma;\Delta)$-strongly balanced with $\Delta^{\rho} \cup \Gamma^{\rho^+} \neq \emptyset$. Then $P \xrightarrow{\tau\langle a,b\rangle}$ with either a or b (+-)responsive name.*

Consider the extension of $\mathrm{wt}(\cdot)$ to system $\vdash_2$, written $\mathrm{wt}^+(\cdot)$, obtained by adding the clause $\mathrm{wt}^+(!\overline{a}\langle b\rangle) = 0$ to the definition of wt. The following proposition is the analog of Proposition 2 for system $\vdash_2$:

Proposition A2 *$\Gamma;\Delta \vdash_2 P$ and $P \xrightarrow{\tau\langle a,b\rangle} P'$ with either a or b (+-)responsive, implies $\mathrm{wt}^+(P') \prec \mathrm{wt}^+(P)$.*

Theorem A1 (Proof of Theorem 5) *Let P be $(\Gamma;\Delta)$-strongly balanced and $r \in \Delta^{\rho} \cup \Gamma^{\rho^+}$. Then P guarantees responsiveness of r.*

Proof. Suppose that $P \xrightarrow{[s]} P'$, with P' having the minimal weight among processes reachable from P with $r \notin s$ (this P' must exist by well-foundness of $\prec$.) Let $s = a_1 \cdots a_n$, and consider the sequence of reductions leading to P':

$$P = P_0 \xrightarrow{[a_1]} P_1 \xrightarrow{[a_2]} \cdots \xrightarrow{[a_n]} P_n = P' \tag{9}$$

By $\Gamma;\Delta \vdash_2 P$ and subject reduction we have that $\Gamma_i;\Delta_i \vdash_2 P_i$ for $i = 0,\ldots,n$, where $\Gamma_0 = \Gamma$ and $\Delta_0 = \Delta$ and $\Gamma_i = \Gamma_{i-1} \ominus^+ (\{a_i\} \setminus \mathrm{in}(P_i))$ and $\Delta_i = \Delta_{i-1} \ominus^+ (\{a_i\} \setminus \mathrm{on}(P_i))$ for $i > 0$. We prove that $P' \xrightarrow{[r]}$ by induction on the number k of non-strongly-balanced processes in the sequence of reductions (9), that is $k = |\{i \,|\, 0 \leq i \leq n \text{ and } P_i \text{ is not } (\Gamma_i,\Delta_i)\text{-strongly balanced }\}|$.

- $k = 0$**:** Then P' is strongly balanced. Since $r \in \Delta_n{}^{\rho} \cup \Gamma_n{}^{\rho^+}$ (as $r \notin s$), by Proposition A1 $P' \xrightarrow{\tau\langle a,b\rangle} P''$, with either a or b (+-)responsive, and by Proposition A2 $\mathrm{wt}^+(P'') \prec \mathrm{wt}^+(P')$. Hence $a = r$, because P' was assumed to have minimal weight among the processes reachable from P without using r as subject.
- $k > 0$**:** Let P_j ($j > 0$) be the leftmost non-strongly-balanced process in the sequence (9). Consider the reduction $P_{j-1} \xrightarrow{[a_j]} P_j$. Since P_{j-1} is strongly balanced while P_j is not, a simple case analysis on the capabilities that a_j may take on shows that $a_j \in (\Gamma_j{}^{\rho^+} \setminus \Delta_j{}^{\rho^+}) \cup (\mathrm{r_i^+}(P) \setminus \mathrm{r_o^+}(P))$, and moreover that a_j must be used as subject of a replicated input in P_{j-1}. Since a_j occurs exactly once in input in P_{j-1}, we must have $P_{j-1} \equiv (\nu\tilde{d})(!a_j(x).R \,|\, S)$ and $P_j \equiv (\nu\tilde{d})(!a_j(x).R \,|\, R[c/x] \,|\, S')$ with $a_j \notin \mathrm{fn}(R[c/x]|S')$. Moreover $P' \equiv (\nu\tilde{d}')(!a(x).R \,|\, P''')$ with $a_j \notin \mathrm{fn}(P''')$. Now, the process $P'_j = (\nu\tilde{d})(R[c/x] \,|\, S')$, obtained by erasing the term $!a_j(x).R$ from P_j, is strongly balanced, and it holds $P'_j \xrightarrow{[a_{j+1}]} \cdots \xrightarrow{[a_n]} P'_n = P''$, with $P'' \equiv (\nu\tilde{d}')P'''$. This sequence has $\leq k-1$ unbalanced processes, and moreover P'' has minimal weight among the processes reachable from P'_j without using r as subject. Then by induction hypothesis $P'' \xrightarrow{[r]}$, which implies $P' \xrightarrow{[r]}$.