A Higher-Order Distributed Calculus with Name Creation

Adrien Piérard and Eijiro Sumii
Graduate School of Information Sciences, Tohoku University
Sendai, Japan

***Abstract*—This paper introduces HOpiPn, the higher-order pi-calculus with *passivation* and *name creation*, and develops an equivalence theory for this calculus. Passivation [Schmitt and Stefani] is a language construct that elegantly models higher-order distributed behaviours like failure, migration, or duplication (e.g. when a running process or virtual machine is copied), and name creation consists in generating a fresh name instead of hiding one. Combined with higher-order distribution, name creation leads to different semantics from name hiding, and is closer to implementations of distributed systems. We define for this new calculus a theory of sound and complete environmental bisimulation to prove reduction-closed barbed equivalence and (a reasonable form of) congruence. We furthermore define environmental *simulations* to prove behavioural *approximation*, and use these theories to show non-trivial examples of equivalence or approximation. Those examples could not be proven with previous theories, which were either unsound or incomplete under the presence of process duplication and name restriction, or else required universal quantification over general contexts.**

***Index Terms*—Higher-order pi-calculus; Distribution and passivation; Name restriction and creation; Environmental bisimulation; Process equivalence**

I. Introduction

Background: With the increasing call for fault tolerance, on-demand computational power and better responsiveness, higher-order and distribution are pervasive in today's computing environment. In this paper, we call *higher-order* the ability to send and receive processes through communication channels, and *distribution* the possibility of location-dependant behaviour. For example, Dell and Hewlett Packard sell products with virtual machine live migration, and Gmail relies on remote execution of JavaScript in the users' browsers. Yet, despite the ubiquity and importance of such higher-order distributed systems, the inherent complexity of these systems makes them difficult to analyse, and thus subject to bugs. Therefore, formal models and methods that help reason about higher-order distribution are sought after.

Passivation [20], [8], [11], [15] is a language abstraction for elegantly modelling higher-order distributed systems in process calculi based on the higher-order π-calculus [16], [18] (with which we assume our reader's familiarity). In its simplest form, passivation consists of a syntax of *located processes* $l[P]$, where l is a name called a *location* and P is a process *located at* l, and two labelled transition rules, $l[P] \xrightarrow{\alpha} l[P']$ if $P \xrightarrow{\alpha} P'$ (TRANSP), and $l[P] \xrightarrow{\overline{l}\langle P\rangle} 0$ (PASSIV), where the relation $P \xrightarrow{\alpha} Q$ in general reads "P does action α and becomes Q." The TRANSP rule states that locations are *transparent*, i.e. do not hide any transition α of the processes they are hosting. The PASSIV rule shows how a located process can be *passivated*, i.e. stopped and output to a channel of the same name as the location.

Despite its simplicity, passivation is yet powerful enough to model complex higher-order distributed behaviours. For example, one can conveniently model failure of a process P at location l as $l[P] \,|\, l(X).\overline{fail} \to 0 \,|\, \overline{fail}$, migration from location l to location m as $l[P] \,|\, l(X).m[X] \to 0 \,|\, m[P]$, or duplication as $l[P] \mid l(X).(l[X] \mid l[X]) \to 0 \mid l[P] \mid l[P]$.

Name creation versus restriction: To our knowledge, previous process calculi with passivation—or, more generally, with higher-order distribution (i.e. communication of processes through channels across locations)—all used so-called *name restriction* [8], [20], [11], [15]. It hides names, forbidding reactions like $\overline{a}.Q \mid \nu a.(a.R_1 \mid \overline{a}.R_2) \to Q \mid \nu a.(R_1 \mid \overline{a}.R_2)$, where the syntax $\nu a.P$ in general means that name a is local to process P, hidden from the outside. Although the name a is textually present in the process $\overline{a}.Q$ above, the a under the ν operator is only visible to $a.R_1$ and $\overline{a}.R_2$, hence not usable for synchronisation with $\overline{a}.Q$.

Nonetheless, sharing of hidden names is possible via name *extrusion*, as in the reaction $l[\nu a.\overline{c}\langle a\rangle.Q] \mid m[c(x).R] \to \nu a.(l[Q]|m[R\{a/x\}])$. This reaction shows that the name a, which was local to location l, can be sent on another channel c and shared with the receiver outside l. In other words, extrusion extends the scope of the sent name to contain the receiver too, possibly crossing location boundaries outwards.

This makes name restriction harder to implement in higher-order distributed settings, as one needs to maintain the scope of extruded names across physically different locations. For example, guaranteeing that the process $\nu a.(l[Q] \,|\, m[R\{a/x\}])$ above cannot interfere with another process that coincidentally uses the same name a seems to require somehow keeping global information about its scope.

By contrast, real implementations of distributed systems often use *name creation* [21], which (perhaps against common belief) leads to a different equivalence theory from that of name restriction. Our name creation consists in forbidding transitions under a ν operator and in generating a fresh name as an internal transition step, shown in the following rule

$$\frac{a \notin s}{s \vdash \nu a.P \xrightarrow{\tau} s \cup \{a\} \vdash P}\textsc{Create}$$

where the syntax $s \vdash Q$ in general reads "process Q, given

a set s of already created names." This rule means that the process $\nu a.P$ can, in an internal transition step, create a name a that is stored immediately in the set of created names, and continue as process P. Assuming that we identify processes up-to alpha-conversion of names bound by the ν operator, the clause $a \notin s$ is a simple precaution to guarantee that freshly created names are indeed unique.

The creationist treatment of names makes the semantics closer to implementations: for example, suppose name creation is the generation of random numbers (arguably unique). Then, two different names in the model would actually be different numbers in the implementation too, thus ruling out interferences between processes and eliminating the need for explicit scope information.

Equivalence of higher-order distributed processes with name creation differs from that with name restriction. Concretely, consider the process

$$P = \nu l.(l[\nu a.(a \mid \overline{a}.\overline{a}.\overline{\omega})] \mid l(X).(X \mid X))$$

which, with name restriction semantics, can at best reduce (in several steps) to $\nu l.(0 \mid \nu a.(0 \mid \overline{a}.\overline{\omega}) \mid \nu a.(0 \mid \overline{a}.\overline{\omega}))$. With name creation semantics, there is also a reduction sequence that leads to the exhibition of name ω by having creation of name a happen *before* passivation and duplication:

$$\begin{array}{lll}
 & \{\omega\} & \vdash \nu l.(l[\nu a.(a \mid \overline{a}.\overline{a}.\overline{\omega})] \mid l(X).(X \mid X)) \\
\rightarrow & \{\omega, l\} & \vdash l[\nu a.(a \mid \overline{a}.\overline{a}.\overline{\omega})] \mid l(X).(X \mid X) \quad \text{(create } l\text{)} \\
\rightarrow & \{\omega, l, a\} & \vdash l[a \mid \overline{a}.\overline{a}.\overline{\omega}] \mid l(X).(X \mid X) \quad \text{(create } a\text{)} \\
\rightarrow & \{\omega, l, a\} & \vdash 0 \mid a \mid \overline{a}.\overline{a}.\overline{\omega} \mid a \mid \overline{a}.\overline{a}.\overline{\omega} \quad \text{(react on } l\text{)} \\
\rightarrow & \{\omega, l, a\} & \vdash 0 \mid 0 \mid \overline{a}.\overline{\omega} \mid a \mid \overline{a}.\overline{a}.\overline{\omega} \quad \text{(react on } a\text{)} \\
\rightarrow & \{\omega, l, a\} & \vdash 0 \mid 0 \mid \overline{\omega} \mid 0 \mid \overline{a}.\overline{a}.\overline{\omega} \quad \text{(react on } a\text{)}
\end{array}$$

Similarly, another (perhaps surprising) difference is the non-bisimilarity between $l[\nu a.\nu b.P]$ and $l[\nu b.\nu a.P]$ with $P = a.b.\ a.\overline{\omega_1} \mid \overline{a}.\overline{b}.\overline{b}.\overline{\omega_2}$, which *are* bisimilar under name restriction. To see it, suppose $s \vdash l[\nu a.\nu b.P]$ with $s = \{l, \omega_1, \omega_2\}$ creates name a and is duplicated (i.e. is passivated and then spawned twice), after which the b of each copy is created, giving $s' \vdash (a.b_1.a.\overline{\omega_1} \mid \overline{a}.\overline{b_1}.\overline{b_1}.\overline{\omega_2}) \mid (a.b_2.a.\overline{\omega_1} \mid \overline{a}.\overline{b_2}.\overline{b_2}.\overline{\omega_2})$ with $s' = s \cup \{a, b_1, b_2\}$. Then, this process can exhibit $\overline{\omega_1}$ using both $\overline{a}$'s and a $\overline{b_1}$, giving $s' \vdash (\overline{\omega_1} \mid \overline{b_1}.\overline{\omega_2}) \mid (a.b_2.a.\ \overline{\omega_1} \mid \overline{b_2}.\overline{b_2}.\overline{\omega_2})$. Yet, it cannot exhibit $\overline{\omega_2}$ which is guarded by two $\overline{b_2}$'s while there is only one b_2. In order for $l[\nu b.\nu a.P]$ to weakly follow and exhibit $\overline{\omega_1}$ too, it must also share a, i.e. create it before duplication, forcing the creation and sharing of b as well. This gives $s \cup \{a, b\} \vdash (a.b.a.\overline{\omega_1} \mid \overline{a}.\overline{b}.\overline{b}.\overline{\omega_2}) \mid (a.\ b.a.\overline{\omega_1} \mid \overline{a}.\overline{b}.\overline{b}.\overline{\omega_2})$ which weakly exhibits not only $\overline{\omega_1}$ but also $\overline{\omega_2}$, therefore telling apart $l[\nu a.\nu b.P]$ and $l[\nu b.\nu a.P]$.

In this paper, we argue that name creation is a realistic alternative to name restriction when modelling higher-order distribution. We recall that a restriction-based semantics is harder to implement, because of the difficulty of implementing distributed scope (which is inherent to the communication of bound names). Here, we discuss several of such semantics and their additional differences from name creation in a higher-order distributed setting. (i) A structural congruence rule $a[\nu c.\ P] \equiv \nu c.a[P]$ (cf. [3]). Under the presence of process duplication, it is "unsound," i.e. makes some inequivalent processes structurally congruent. For example, it allows $a[\nu c.(\overline{c}|c.c.\ \bar{\omega})] \equiv \nu c.a[\overline{c}|c.c.\bar{\omega}]$, but the two processes are distinguished by an observer $R = a(X).(X \mid X)$. (ii) Enforcing extrusion [20] before passivation like $l[\nu c.P] \mid l(X).(m[X] \mid n[X]) \rightarrow \nu c.\ (m[P] \mid n[P])$. It does not allow duplication without sharing private channels, keeping passivation from being used as a general device for copying. (iii) Forbidding passivation when ν is in evaluation position, i.e. $l[\nu c.P] \stackrel{\overline{l}\langle \nu c.P\rangle}{\not\rightarrow} 0$. It hinders duplication with private channels as well. Moreover, expectedly equivalent processes $l[\nu a.a[P]]$ and $l[P]$ are distinguished by $l(X).\overline{\omega}$, which reacts only with $l[P]$. (iv) Vertical extrusion with an extra rule like $l[\nu c.P] \stackrel{\tau}{\rightarrow} \nu c.l[P]$. It differs from name creation too: consider $Q = l[m[\nu a.P] \mid m(X).(X \mid X)] \mid l(Y).\ (Y \mid Y)$ which can become $Q' = l[\nu a.m[P] \mid m(X).(X \mid X)] \mid l(Y).(Y \mid Y)$ in a step, and then (weakly) become either $\nu a.\ (P \mid P \mid P \mid P)$ with one bound name, or $\nu a.(P \mid P) \mid \nu a.(P \mid P)$ with two. With name creation, there is no reduction $Q \Rightarrow Q'$ such that only $Q' \Rightarrow P\{a_1/a\} \mid P\{a_1/a\} \mid P\{a_1/a\} \mid P\{a_1/a\}$ and $Q' \Rightarrow (P\{a_1/a\} \mid P\{a_1/a\}) \mid (P\{a_2/a\} \mid P\{a_2/a\})$ with a_1 and a_2 fresh, whence the difference.

Equivalence and inequivalence in higher-order distribution: We have just seen that equivalence differs depending on the semantics of names. Consequently, the equivalence theory under the presence of name creation needs to be rethought. Behavioural equivalence can be characterised as reduction-closed barbed equivalence (or congruence) [9] which has a simple definition but is impractical as a proof method because of a universal quantification on observer processes (or contexts) in its definition. Therefore, more convenient relations like bisimulations, whose membership implies reduction-closed barbed equivalence and which come with a co-inductive proof method, are sought after.

Accordingly, we define a theory of (environmental [22], [23], [17], [19], [15]) *bisimulation* for a higher-order π-calculus with both passivation and name creation. The theory is proven to be sound and, thanks to name creation semantics, complete. (In contrast, environmental bisimulations for higher-order π-calculus with passivation [15] were far from complete under name restriction semantics, proven sound only with severe constraints on environments.) It can then be used to prove non-trivial equivalences that could not be shown previously [15], [11], like that of distributed left and right list folds (simplified versions of "MapReduce", detailed in Section V.

One may also want to prove bisimilarity of distributed programs that are more structurally different than the two fold functions, e.g. linear and logarithmic implementations of power functions. Perhaps surprisingly again, these implementations are *not* bisimilar. The reason is that the linear distributed implementation uses more hosts than the logarithmic one, and is therefore more likely to fail (under either of name creation and restriction). Thus, bisimilarity may sometimes be too strong an equivalence. Instead, mutual *simulation* can be desirable, so as to provide a coarser equivalence (cf. [12, p. 20, Exercise 3.10]) still useful for comparing such programs. En-

vironmental simulations can be proven, for example, between distributed linear and logarithmic power functions as detailed in Section VI-A.

Summary of our contributions: In this paper, we introduce the higher-order π-calculus with passivation and name creation (henceforth HOπPn), a dialect of HOπP (the higher-order π-calculus with passivation and name restriction) [11]. We then provide environmental bisimulations that are sound and complete with respect to reduction-closed barbed equivalence and (a reasonable form of) congruence, and use them to prove a non-trivial equivalence that could not be shown with previous methods. We also provide sound environmental simulations that can be used to show reduction-closed barbed *approximation*, and give a non-trivial simulation proof as well.

Overview of the paper: The rest of the paper is structured as follows. Section II defines HOπPn. Section III formalises our environmental bisimulations and simulations, and Section IV establishes their soundness and completeness. Section V shows the example bisimilarity proof of distributed left and right folds. Section VI discusses non-bisimilar examples and their mutual simulation proofs. Finally, Section VII considers previous and future work.

II. Higher-Order π-Calculus with Passivation and Name Creation

We formally introduce HOπPn through its syntax and labelled transition system. The syntax of HOπPn processes P, Q and terms M, N is given by the following grammar (the same as in [15]):

$$\begin{array}{rcl} P, Q & ::= & 0 \mid a(X).P \mid \overline{a}\langle M\rangle.P \mid (P \mid Q) \mid a[P] \\ & \mid & \nu a.P \mid\ !P \mid \mathit{run}(M) \\ M, N & ::= & X \mid\ `P \end{array}$$

Briefly, X ranges over the set of variables, and a over names. 0 is a stuck process, $a(X)$ and $\overline{a}\langle M\rangle$ input and output prefixes, $\mid$ the parallel composition operator, and $a[P]$ the process P located at location a. νa is the name creation prefix, ! the replication operator, *run* the thawing operator which is used to turn a term into a process, X a variable and $`P$ a process as a term. As in [15], the distinction between processes and terms is needed for our generic up-to context technique (see Section III).

The semantics of HOπPn is given by the following labelled transitions system which is based on that of the higher-order π-calculus with passivation [11]—itself being based on that of the higher-order π-calculus [16]—and is now defined on *configurations*. A configuration $s \vdash P$ is the pair of a set s of names and a process P such that $\mathit{fn}(P) \subseteq s$. We casually write sx or s, x for $s \cup \{x\}$ or $s \cup x$ when x is a name or a set of names. Omitting symmetric rules Par-R and React-R, the transition relation is defined inductively by the rules in Fig. 1. Assuming knowledge of the standard higher-order π-calculus [18], [16], we skim over the distribution-related transitions and comment on the notable changes coming from name creation. The Transp rule expresses the *transparency* of locations—the fact that transitions can happen inside a location

$$\frac{}{s \vdash a(X).P \xrightarrow{a(M)} s \vdash P\{M/X\}}\textsc{Ho-In}$$

$$\frac{}{s \vdash \overline{a}\langle M\rangle.P \xrightarrow{\overline{a}\langle M\rangle} s \vdash P}\textsc{Ho-Out}$$

$$\frac{s \vdash P \xrightarrow{\alpha} s' \vdash P' \quad (s' \setminus s) \cap \mathit{fn}(Q) = \emptyset}{s \vdash P \mid Q \xrightarrow{\alpha} s' \vdash P' \mid Q}\textsc{Par-L}$$

$$\frac{s \vdash P \xrightarrow{\overline{a}\langle M\rangle} s \vdash P' \quad s \vdash Q \xrightarrow{a(M)} s \vdash Q'}{s \vdash P \mid Q \xrightarrow{\tau} s \vdash P' \mid Q'}\textsc{React-L}$$

$$\frac{s \vdash !P \mid P \xrightarrow{\alpha} s' \vdash P'}{s \vdash !P \xrightarrow{\alpha} s' \vdash P'}\textsc{Rep} \qquad \frac{a \notin s}{s \vdash \nu a.P \xrightarrow{\tau} s, a \vdash P}\textsc{Create}$$

$$\frac{s \vdash P \xrightarrow{\alpha} s' \vdash P}{s \vdash a[P] \xrightarrow{\alpha} s' \vdash a[P']}\textsc{Transp}$$

$$\frac{}{s \vdash a[P] \xrightarrow{\overline{a}\langle `P\rangle} s \vdash 0}\textsc{Passiv} \qquad \frac{}{s \vdash \mathit{run}(`P) \xrightarrow{\tau} s \vdash P}\textsc{Run}$$

Fig. 1. Labelled transitions system of HOπPn

and be observed outside its boundaries. The Passiv rule shows how a process running inside a location can be *passivated*, i.e. stopped, turned into a term, and sent along a channel whose name corresponds to that of the location. The Run rule shows how to retrieve a process from a term at the cost of an internal transition.

The rule Create shows how a process $\nu a.P$ can create a name a—which is added to the configuration's set of names—and become P in an internal transition step. As we identify processes up-to alpha-conversion of bound names, progress is guaranteed. The rule Par-L shows that a transition can happen in a sub-process provided it does not create a name that is free in another sub-process put in parallel (the function fn, which returns the set of free names of a process or a term, is standard). Again, alpha-conversion is used for guaranteeing that no free name of Q will be captured.

The other rules are straightforward even in the presence of name creation and will not be discussed further. As usual with small-step semantics, when the assumptions cannot be satisfied or when a case is undefined (as in $\mathit{run}(X)$), transition does not progress and the process is stuck.

Henceforth, we shall write $\overline{a}.P$ for $\overline{a}\langle 0\rangle.P$ and $a.P$ for $a(X).P$ when X is not free in P. We define structural congruence $\equiv$ as the smallest congruence on processes with $P \equiv P \mid 0$, $P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$, $P_1 \mid P_2 \equiv P_2 \mid P_1$ and $!P \equiv !P \mid P$. Notice that this definition is *not* standard: it allows neither $(\nu a.P) \mid Q \equiv \nu a.(P \mid Q)$, $\nu a.\nu b.P \equiv \nu b.\nu a.P$, nor $\nu a.0 \equiv 0$.

III. Environmental Bisimulation and Simulation for HOπPn

We define an *environmental relation* $\mathcal{X}$ as a set of sextuples $(\mathcal{E},\ r,\ s,\ P,\ t,\ Q)$ where $\mathcal{E}$ is a binary relation (called the *environment*) on terms with no free variables and finitely many free names, r is a finite set of names (the *public* names), s and t too are finite sets of names (the *created* names) such that $r \subseteq s \cap t$, and P and Q are variable-closed processes

(the *tested processes*). We often write $(s \vdash P)\,\mathcal{X}_{\mathcal{E};r}\,(t \vdash Q)$ to mean $(\mathcal{E}, r, s, P, t, Q) \in \mathcal{X}$ for an environmental relation $\mathcal{X}$.

Definition 1. *We define multi-hole contexts for terms C (contexts that have holes for terms) and multi-hole contexts for processes C_p (contexts that have holes for processes) as:*

$$C ::= 0 \mid a(X).C \mid \overline{a}\langle D\rangle.C \mid (C \mid C) \mid a[C] \mid \nu a.C \mid\, !C \mid \mathit{run}(D)$$
$$D ::= [\cdot]_i \mid X \mid \text{`}C$$
$$C_p ::= [\cdot]_i \mid 0 \mid a(X).C_p \mid \overline{a}\langle D_p\rangle.C_p \mid (C_p \mid C_p) \mid a[C_p] \mid \nu a.C_p \mid\, !C_p \mid \mathit{run}(D_p)$$
$$D_p ::= X \mid \text{`}C_p$$

Definition 2. *We define* process context closure *and* term context closure *as:*

$$(\mathcal{E};r)^\circ = \{(C[\widetilde{M}], C[\widetilde{N}]) \mid bn(C) \cap fn(\widetilde{M},\widetilde{N}) = \emptyset, fn(C) \subseteq r,\ (\widetilde{M},\widetilde{N}) \in \mathcal{E}\}$$
$$(\mathcal{E};r)^\star = \{(D[\widetilde{M}], D[\widetilde{N}]) \mid bn(D) \cap fn(\widetilde{M},\widetilde{N}) = \emptyset, fn(D) \subseteq r,\ (\widetilde{M},\widetilde{N}) \in \mathcal{E}\}$$

The process context closure $(\mathcal{E};r)^\circ$ intuitively represents the processes that an attacker can craft given some terms and public names. It allows him to create processes using terms $(\widetilde{M},\widetilde{N})$ from the environment and names from r. Capture of names is forbidden, hence the condition on names bound by the context. As this closure uses a context for terms, it will necessarily put its terms in an output prefix or under a *run*. The term context closure $(\mathcal{E};r)^\star$ intuitively corresponds to all the terms that the attacker can craft from his knowledge. We point out that these closure operations are monotonic on all their arguments, and thus for any $\mathcal{E}$ and r, $(\mathcal{E};r)^\star$ includes the identity $(\emptyset;r)^\star$.

A few extra notations are used in this paper. We define the weak transitions $\stackrel{\tau}{\Rightarrow}$ (or $\Rightarrow$) as the reflexive transitive closure of $\stackrel{\tau}{\rightarrow}$ (or $\rightarrow$), and $\stackrel{\alpha}{\Rightarrow}$ as $\stackrel{\tau}{\Rightarrow}\stackrel{\alpha}{\rightarrow}\stackrel{\tau}{\Rightarrow}$ for any $\alpha \neq \tau$. Finally, we write $a \oplus b$ to express the union $\{a\} \cup b$.

We now formally define environmental bisimulations (a subset of environmental relations):

Definition 3. *$\mathcal{X}$ is an environmental bisimulation if for all $(s \vdash P)\,\mathcal{X}_{\mathcal{E};r}\,(t \vdash Q)$,*

1) *if $s \vdash P \stackrel{\tau}{\rightarrow} s' \vdash P'$ then there is $t' \vdash Q'$ such that $t \vdash Q \stackrel{\tau}{\Rightarrow} t' \vdash Q'$ and $(s' \vdash P')\,\mathcal{X}_{\mathcal{E};r}\,(t' \vdash Q')$,*
2) *if $s \vdash P \xrightarrow{a(M)} s \vdash P'$ with $a \in r$ and $(M,N) \in (\mathcal{E};r)^\star$, then there is $t' \vdash Q'$ such that $t \vdash Q \stackrel{a(N)}{\Longrightarrow} t' \vdash Q'$ and $(s \vdash P')\,\mathcal{X}_{\mathcal{E};r}\,(t' \vdash Q')$,*
3) *if $s \vdash P \xrightarrow{\overline{a}\langle M\rangle} s \vdash P'$ with $a \in r$, then there are $t' \vdash Q'$ and N such that $t \vdash Q \stackrel{\overline{a}\langle N\rangle}{\Longrightarrow} t' \vdash Q'$ and $(s \vdash P')\,\mathcal{X}_{(M,N)\oplus\mathcal{E};r}\,(t' \vdash Q')$,*
4) *for all $l \in r$ and $(\text{`}P_1, \text{`}Q_1) \in \mathcal{E}$, we have $(s \vdash P \mid l[P_1])\,\mathcal{X}_{\mathcal{E};r}\,(t \vdash Q \mid l[Q_1])$,*
5) *for all $n \notin s \cup t$, we have $(s, n \vdash P)\,\mathcal{X}_{\mathcal{E};r,n}\,(t, n \vdash Q)$, and*
6) *the converse of the three first clauses, on Q's transitions.*

Clause 1 requires weak reduction closure and is fairly usual; clause 2 requires tested processes in a bisimulation to be able to input on a public channel any related terms that the attacker may create (hence the use of the term context closure), and to have their continuations in the bisimulation; clause 3 enlarges the knowledge of the attacker with terms that were output on a public channel, and requires the continuations to be bisimilar under this new knowledge; clause 4 allows the attacker to spawn and immediately run terms from the environment as processes in parallel to the tested processes (this allows to virtually consider an arbitrary process from the process context closure, while being much more tractable [15, Section 1]); clause 5 means that the attacker can create fresh names at will; finally, clause 6 is just the symmetric of the first three ones.

We remind that, because the set r of names is included in both s and t, we know that no name created by P nor Q will clash with r; this is why we do not need extra constraints on names like $(s' \setminus s) \cap r = \emptyset$ (in clause 1) and $(t' \setminus t) \cap r = \emptyset$ (in clauses 1, 2, 3), and why we do not have to require $n \notin r$ in clause 5. Also, using clause 5, the attacker can always generate fresh names before creating terms (that will use these new names) for input in clause 2.

As all the clauses of environmental bisimulations are monotonic on $\mathcal{X}$, the union of all bisimulations exists and is itself an environmental bisimulation. We call it environmental bisimilarity and write it $\sim$. For proving the equivalence of two processes P and Q, we show that $(f \vdash P) \sim_{\emptyset;r} (f \vdash Q)$ for some $r \subseteq f = fn(P,Q)$. It corresponds to equivalence where the attacker can send and receive messages over the public channels r of P and Q, but is yet to learn and put any term into the environment. Since $\sim$ is the union of all bisimulations, to prove this equivalence, it suffices to find an environmental bisimulation $\mathcal{X}$ such that $(f \vdash P)\,\mathcal{X}_{\emptyset;r}\,(f \vdash Q)$ with public names $r \subseteq f = fn(P,Q)$.

For improving the practicality of our proof method, we define an up-to context technique. Let us write $\mathcal{X}^\star$ for an environmental relation $\mathcal{X}$, such that:

$$\begin{aligned}\mathcal{X}^\star = \{(\mathcal{E}, r, s, P, t, Q) \mid\ & P \equiv P_0 \mid P_1,\ Q \equiv Q_0 \mid Q_1,\\ & r' \cap (s \cup t) = \emptyset,\\ & (s, r' \vdash P_0)\,\mathcal{X}_{\mathcal{E}';r,r'}\,(t, r' \vdash Q_0),\\ & (P_1, Q_1) \in (\mathcal{E}'; rr')^\circ,\ \mathcal{E} \subseteq (\mathcal{E}'; rr')^\star\}\end{aligned}$$

Even though we call it "up-to context" for simplicity, it is in fact the combination of several up-to techniques: (i) "up-to context" since we allow the spawning of any related processes (P_1, Q_1) taken from the process context closure $(\mathcal{E}'; rr')^\circ$ of the knowledge $\mathcal{E}'$, rr' in parallel to the tested processes P_0 and Q_0 related by $(s, r' \vdash \cdot)\mathcal{X}_{\mathcal{E}';rr'}(t, r' \vdash \cdot)$; (ii) "up-to environment" since we allow, through the condition with the term context closure, the use of environments that are *larger* than immediately necessary; (iii) "up-to name creation" since we allow the use of extra new names r'; and (iv) "up-to structural congruence" since we identify processes structurally congruent to $P_0 \mid P_1$ and $Q_0 \mid Q_1$. This convenient notation allows us to define environmental bisimulations up-to context:

Definition 4. *$\mathcal{X}$ is an environmental bisimulation up-to context if for all $(s \vdash P)\,\mathcal{X}_{\mathcal{E};r}\,(t \vdash Q)$,*

1) *if* $s \vdash P \xrightarrow{\tau} s' \vdash P'$ *then there is* $t' \vdash Q'$ *such that* $t \vdash Q \stackrel{\tau}{\Rightarrow} t' \vdash Q'$ *and* $(s' \vdash P')\ \mathcal{X}^{\star}_{\mathcal{E};r}\ (t' \vdash Q')$,
2) *if* $s \vdash P \xrightarrow{a(M)} s \vdash P'$ *with* $a \in r$ *and* $(M, N) \in (\mathcal{E}; r)^{\star}$, *then there is* $t' \vdash Q'$ *such that* $t \vdash Q \stackrel{a(N)}{\Longrightarrow} t' \vdash Q'$ *and* $(s \vdash P')\ \mathcal{X}^{\star}_{\mathcal{E};r}\ (t' \vdash Q')$,
3) *if* $s \vdash P \xrightarrow{\overline{a}\langle M\rangle} s \vdash P'$ *with* $a \in r$, *then there are* $t' \vdash Q'$ *and* N *such that* $t \vdash Q \stackrel{\overline{a}\langle N\rangle}{\Longrightarrow} t' \vdash Q'$ *and* $(s \vdash P')\ \mathcal{X}^{\star}_{(M,N)\oplus\mathcal{E};r}\ (t' \vdash Q')$,
4) *for all* $l \in r$ *and* $(`P_1, `Q_1) \in \mathcal{E}$, *we have* $(s \vdash P \mid l[P_1])\ \mathcal{X}^{\star}_{\mathcal{E};r}\ (t \vdash Q \mid l[Q_1])$,
5) *for all* $n \notin s \cup t$, *we have* $(s, n \vdash P)\ \mathcal{X}_{\mathcal{E};r,n}\ (t, n \vdash Q)$, *and*
6) *the converse of the three first clauses, on* Q*'s transitions.*

This is basically the same definition as Definition 3 but all the positive instances of $\mathcal{X}$ became $\mathcal{X}^{\star}$ (except in clause 5 for technical reasons). Clause 4 is not a tautology since it spawns terms immediately as processes while the definition of $\mathcal{X}^{\star}$ allows only quoted processes. This distinction between quoted and non-quoted processes enables the use of generic contexts (as in [19], [15]) instead of redex contexts (as in [17]). Similarly to $\sim$, we define environmental bisimilarity up-to context and write it $\simeq$.

Finally, we define environmental similarity $\prec$ and similarity up-to context $\preceq$ by removing the converse conditions from the appropriate definitions.

IV. Soundness and Completeness of Environmental Bisimulation and Simulation

We outline here main results and proofs concerning the soundness and completeness of our proof method. More details are found in the appendix [14].

A. Behavioural Equivalences

We say process P has or exhibits barb a (resp. $\overline{a}$), written $P \downarrow_a$ (resp. $P \downarrow_{\overline{a}}$), whenever $P \xrightarrow{a(\cdot)} \cdot$ (resp. $P \xrightarrow{\overline{a}\langle\cdot\rangle} \cdot$). We say process P weakly exhibits barb μ, written $P \Downarrow \mu$, whenever $P \Rightarrow \downarrow \mu$ for a name or a co-name μ.

We can now formally define the equivalence predicates of our language, based on that of [9] (see also [18, Section 2.4.4]) with extensions for name creation.

Definition 5. *Reduction-closed barbed equivalence* $\approx$ *is the largest binary relation on variable-closed configurations, indexed with a set of names* $r \subseteq s \cap t$, *such that when* $s \vdash P \approx_r t \vdash Q$,

- $s \vdash P \to s' \vdash P'$ *implies there are* Q' *and* t' *such that* $t \vdash Q \Rightarrow t' \vdash Q'$ *and* $s' \vdash P' \approx_r t' \vdash Q'$,
- $s \vdash P \downarrow_\mu$ *implies* $t \vdash Q \Downarrow_\mu$ *if* μ *or* $\overline{\mu}$ *is in* r,
- *the converse of the above two on* Q, *and*
- *for all* R *with* $fn(R) \cap ((s \cup t) \setminus r) = \emptyset$, *we have* $s \cup fn(R) \vdash P \mid R \approx_{r \cup fn(R)} t \cup fn(R) \vdash Q \mid R$.

Note that we parameterised the equivalence with public names r. This is necessary for distinguishing the public names from private names that are not known to the attacker and cannot be observed nor used. This explains why the clause on barbs (and its symmetric) only considers barbs in r, and why in the last clause private names cannot be free in R. However, the free names created by the attacker are public and must thus be added to r for observation, and to s and t to avoid re-creation.

Definition 6. *Reduction-closed barbed congruence* $\dot{\approx}$ *is defined similarly to Definition 5, but replacing* $\approx$ *with* $\dot{\approx}$ *and the last clause with: for all* C_p *(context with holes for processes) such that* $fn(C_p) \cap ((s \cup t) \setminus r) = bn(C_p) \cap fn(P, Q) = \emptyset$, *we have* $s \cup fn(C_p) \vdash C_p[P] \dot{\approx}_{r \cup fn(C_p)} t \cup fn(C_p) \vdash C_p[Q]$.

It might be surprising that we consider a "congruence" which cannot capture public names ($bn(C_p) \cap fn(P, Q) = \emptyset$), but we argue that this is a reasonable definition. Indeed, free names in our language represent already created constants (private or public) in the compared processes; allowing the capture of names would virtually correspond to allowing in-place changes to the constant values in programs. Even though this may well tell some systems apart—as the attacker wishes to do—we doubt it represents a reasonable way to compare the behaviours of systems in execution contexts (in fact, this rather looks like using a binary editor to tell apart programs by modifying their code).

Definition 7. *Reduction-closed barbed approximation* $\lessapprox$ *is defined similarly to Definition 5, but replacing* $\approx$ *with* $\lessapprox$ *and removing the converse clauses. Respectively, reduction-closed barbed pre-congruence* $\dot{\lessapprox}$ *is defined similarly to Definition 6, but replacing* $\dot{\approx}$ *with* $\dot{\lessapprox}$ *and removing the converse clause.*

We say P approximates Q if $f \vdash P \lessapprox_r f \vdash Q$ with some $r \subseteq f = fn(P, Q)$. Intuitively, $P \lessapprox Q$ (or $P \dot{\lessapprox} Q$) whenever Q can do at least as much as P, in parallel with an observer R (or under a non-capturing context C_p).

B. Soundness

Theorem 1. *If* $(s \vdash P) \simeq_{\mathcal{E};r} (t \vdash Q)$ *then* $(s \vdash P) \sim_{\mathcal{E};r} (t \vdash Q)$.

Outline of proof: knowing $\simeq \subseteq \simeq^{\star}$ by definition, we show that:

1) transitions from $\simeq^{\star}$ lead to $\simeq^{-}$ (a superset of $\simeq^{\star}$ called run-erasure [14]),
2) if $(s \vdash P) \simeq^{-}_{\mathcal{E};r} (t \vdash Q)$ and $s \vdash P \xrightarrow{\overline{a}\langle M\rangle} s \vdash P'$, then $t \vdash Q \stackrel{\overline{a}\langle N\rangle}{\Longrightarrow} t' \vdash Q'$ and $(s \vdash P') \simeq^{-}_{(M,N)\oplus\mathcal{E};r} (t' \vdash Q')$, and using this result,
3) $\simeq^{-}$ is also closed by input and internal transitions.

It is then quite easy to show that $\simeq^{-}$ verifies all the clauses of environmental bisimulations, that is, $\simeq^{-} \subseteq \sim$. □

Corollary 1. *If* $(f \vdash P) \sim_{\emptyset;r} (f \vdash Q)$ *with* $r \subseteq f = fn(P, Q)$, *then* $f \vdash P \approx_r f \vdash Q$.

Outline of proof: we show that $\sim$ is reduction-closed (by definition), that it weakly exhibits the same barbs (by definition of bisimulation, ignoring the continuations after input or output

transitions), and that it is preserved by parallel composition of arbitrary processes (that do not use private names) using the up-to context technique (with $\sim \subseteq \simeq$). □

It is interesting to remark that reduction-closed barbed *congruence* can easily be shown as follows. Let us define $P \mathrel{\dot{=}}_r Q$ if $(f \vdash 0) \sim_{\{(`P,`Q)\};r} (f \vdash 0)$ with $r \subseteq f = fn(P,Q)$. Then:

Theorem 2. *If $P \mathrel{\dot{=}}_r Q$, then $f \vdash P \mathrel{\dot{\approx}}_r f \vdash Q$ with $r \subseteq f = fn(P,Q)$.*

Outline of proof: We first show that a set relating run-erasures of $(C[\widetilde{M}], C[\widetilde{N}])$ for any non-capturing context C, with $(s \vdash 0) \sim_{\mathcal{E};r} (t \vdash 0)$ and $(\widetilde{M}, \widetilde{N}) \in \mathcal{E}$, is reduction-closed and verifies the conditions on barbs of reduction-closed barbed congruence. Then, by $P \mathrel{\dot{=}}_r Q$, i.e. $(f \vdash 0) \sim_{\{(`P,`Q)\};r} (f \vdash 0)$ with $r \subseteq f = fn(P,Q)$, we have $f \vdash P \mathrel{\dot{\approx}}_r f \vdash Q$. □

We emphasise that a capturing congruence cannot (and should not) be shown with this method. Omitting ‘ and *run* for brevity, we prove this by crafting a counter-example such that $P \mathrel{\dot{=}}_r Q$ but P and Q are not related by a name-capturing version of $\mathrel{\dot{\approx}}$.

Let $P_1 = a(X).(X \mid i(Y).\overline{m}) \mid m.\overline{\omega}$ and $Q_1 = a(X).(X \mid i(Y).Y) \mid m.\overline{\omega}$. We then consider the two processes $P = \nu i.\overline{b}\langle P_1\rangle \mid \overline{c}\langle\overline{a}\langle\overline{i}\langle\overline{m}\rangle\rangle\rangle$ and $Q = \nu i.\overline{b}\langle Q_1\rangle \mid \overline{c}\langle\overline{a}\langle\overline{i}\langle\overline{m}\rangle\rangle\rangle$ which are such that $P \mathrel{\dot{=}}_r Q$ for $r = \{a, m, b, c, \omega\}$. P and Q have been designed such that P can exhibit barb $\overline{\omega}$ if it receives anything on private channel i, while Q can exhibit barb $\overline{\omega}$ if it receives process $\overline{m}$ on private channel i.

By creating name i and then capturing the public name m with a context like $\overline{a}\langle\nu m.[\,]_1\rangle \mid a(X).(X \mid X)$, it is possible to reach a state where $\overline{\omega}$ and $\overline{m_1} \mid m_2.\overline{\omega}$ with private and different m_1 and m_2 would be related. However, as only the former process has barb $\overline{\omega}$, the equivalence cannot hold. This shows that comparing processes in a bisimulation environment is not enough to guarantee name-capturing reduction-closed barbed congruence.

It is in fact no problem that two processes in a bisimulation environment are not necessarily related by a name-capturing version of reduction-closed barbed congruence. Indeed, it is consistent with our idea that allowing capture of already created names is not a good basis for a congruence in languages with name creation like HOπPn.

Soundness also holds for simulations:

Theorem 3. *Let $r \subseteq f = fn(P,Q)$. If $(f \vdash P) \preceq_{\emptyset;r} (f \vdash Q)$, then $(f \vdash P) \lesssim_r (f \vdash Q)$. Respectively, if $(f \vdash 0) \preceq_{\{(`P,`Q)\};r} (f \vdash 0)$, then $(f \vdash P) \mathrel{\dot{\lesssim}}_r (f \vdash Q)$.*

The proof is immediate as our soundness proofs for environmental bisimulations do not use the symmetry condition and therefore can automatically be applied to environmental simulations too.

C. Completeness

Theorem 4. *If $f \vdash P \approx_r f \vdash Q$ with $r \subseteq f = fn(P,Q)$, then $(f \vdash P) \sim_{\emptyset;r} (f \vdash Q)$.*

Outline of proof: we find an environmental bisimulation $\mathcal{X}$ (up-to context) relating reduction-closed barbed equivalent P and Q. The trick is to use a parallel product of outputting processes to represent the environment. Roughly,

$$(s \vdash P \mid \textstyle\prod_i l_i[P_i])\ \mathcal{X}_{\mathcal{E};r}\ (t \vdash Q \mid \prod_i l_i[Q_i])$$

with $\{(`\widetilde{P}, `\widetilde{Q})\} \subseteq \mathcal{E}$ and $\widetilde{l} \in r$ is defined by

$$(s, \widetilde{g} \vdash P \mid \textstyle\prod_j\, !\overline{g_j}\langle `P_j\rangle) \approx_{r,\widetilde{g}} (t, \widetilde{g} \vdash Q \mid \prod_j\, !\overline{g_j}\langle `Q_j\rangle)$$

with $\{(`\widetilde{P}, `\widetilde{Q})\} = \mathcal{E}$. By using the last clause of reduction-closed barbed equivalence, one can create processes that will fetch the necessary $(`\widetilde{P}, `\widetilde{Q})$ and use them for crafting elements of the context closure $(\mathcal{E};r)^\star$ needed in the input clause. The spawn clause is satisfied by construction. When accounting for a reaction like $P \mid l_1[P_1] \xrightarrow{\tau} P \mid l_1[P_1']$, one uses the last clause of reduction-closed barbed equivalence to create a receiver $l_1[g_1(X).X]$, spawns P_1 (and Q_1) inside this location l_1, mimics the reduction of P_1 (and the weak reactions of Q and Q_1 to Q' and Q_1'), and then passivates the contents of location l_1 to put $(`P_1', `Q_1')$ immediately in the representation of the "environment" under fresh name g_{j+1}, giving

$$s', \widetilde{g} \vdash P \mid \textstyle\prod_j\, !\overline{g_j}\langle `P_j\rangle \mid\, !\overline{g_{j+1}}\langle `P_1'\rangle \approx_{r,\widetilde{g}}$$
$$t', \widetilde{g} \vdash Q' \mid \textstyle\prod_j\, !\overline{g_j}\langle `Q_j\rangle \mid\, !\overline{g_{j+1}}\langle `Q_1'\rangle.$$

Therefore, processes $P \mid l_1[P_1']$ and $Q' \mid l[Q_1']$ are now related as wanted. □

Corollary 2. *If $f \vdash P \mathrel{\dot{\approx}}_r f \vdash Q$ with $r \subseteq f = fn(P,Q)$, then $P \mathrel{\dot{=}}_r Q$.*

Outline of proof: by the last clause of reduction-closed barbed congruence, we know that $(f \vdash P) \mathrel{\dot{\approx}}_r (f \vdash Q)$ implies $(a, f \vdash \overline{a}\langle P\rangle) \mathrel{\dot{\approx}}_{a,r} (a, f \vdash \overline{a}\langle Q\rangle)$ for fresh a, which itself implies $(a, f \vdash \overline{a}\langle P\rangle) \approx_{a,r} (a, f \vdash \overline{a}\langle Q\rangle)$. By Theorem 4, we thus have $(a, f \vdash \overline{a}\langle P\rangle) \sim_{\emptyset;a,r} (a, f \vdash \overline{a}\langle Q\rangle)$. We then output to a, get $(a, f \vdash 0) \sim_{\{(`P,`Q)\};a,r} (a, f \vdash 0)$, remove a up-to name creation, and we are done. □

We do not know yet whether completeness of simulation holds, since our current proofs rely on the symmetry conditions of the relations.

V. Bisimilarity Example

We present an example of equivalence that could not be proven with previous methods, remotely inspired by MapReduce and abstracting the "reduce" part of it. More precisely, we show the bisimilarity between distributed left- and right-fold computations for arbitrary list l, associative function f, and initial value i (the identity element of f). With *car* and *cdr* functions that return the head and tail of a list, we define the "fold servers" as:

$$L = !\mathit{fl}(f,i,l,k).\mathsf{if}\ \mathit{null}\ l$$
$$\quad \mathsf{then}\ \overline{k}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{\mathit{fl}}\langle f, f\ i\ (\mathit{car}\ l), \mathit{cdr}\ l, c\rangle.c(m).\overline{k}\langle m\rangle$$
$$R = !\mathit{fr}(f,l,i,k).\mathsf{if}\ \mathit{null}\ l$$
$$\quad \mathsf{then}\ \overline{k}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{\mathit{fr}}\langle f, \mathit{cdr}\ l, i, c\rangle.c(m).\overline{k}\langle f\ (\mathit{car}\ l)\ m\rangle$$

They are parameterised by (in addition to f, l and i) a channel k to return their results to clients (although omitted in the

$$\mathcal{X}_1 = \{(\emptyset,\ r,\ r\,fl,\ P,\ r\,fr,\ Q)\ \big|\ \{a,b\} \subseteq r,\ fl, fr \notin r\}$$
$$\mathcal{X}_2 = \{(\mathcal{E},\ r,\ r\,fl,\ \overline{fl}\langle f,i,l,k\rangle \mid \textstyle\prod_{j=1}^{n} a_j[L],\ r\,fr,\ \overline{fr}\langle f,l,i,k\rangle \mid \prod_{j=1}^{n} a_j[R])\ \big|\ \mathcal{E} = \{(`L, `R)\},\ \{k,a,b,a_1,\ldots,a_n\} \subseteq r,\ fl, fr \notin r\}$$
$$\mathcal{X}_3 = \{(\mathcal{E}, r, r\,fl\,\widetilde{c}, \textstyle\prod_j a_j[P_j], r\,fr\,\widetilde{c}, \prod_j a_j[Q_j])\ \big|\ \mathcal{E} = \{(`L \mid \prod_h A_h,\ `R \mid \prod_h B_h)\ \big|\ (\widetilde{A},\widetilde{B}) \in E^{f,l,i}(\mathsf{length}\ l,\ \{k\},\ r\,fl\,fr)\},\ \{k,a,b,\widetilde{a}\} \subseteq r,\ \{\widetilde{c}\} = fn(\mathcal{E}) \setminus r \setminus \{fl, fr\},\ (`\widetilde{P}, `\widetilde{Q}) \in \mathcal{E},\ fl, fr \notin r\}$$

$$E^{f,l,i}(0,\ rep,\ cre) = \{$$
$$(\mathsf{if}\ null\ []\ \mathsf{then}\ \overline{c_0}\langle f_d^l\rangle\ \mathsf{else}\ \nu c.\overline{fl}\langle f, f\ f_d^l\ (car\ []), cdr\ [], c\rangle.c(m).\overline{c_0}\langle m\rangle,$$
$$\mathsf{if}\ null\ []\ \mathsf{then}\ \overline{c_0}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{fr}\langle f, cdr\ [], i, c\rangle.c(m).\overline{c_0}\langle f\ (car\ [])\ m\rangle),$$
$$(\overline{c_0}\langle f_d^l\rangle,\ \overline{c_0}\langle i\rangle)\ \ \big|\ \ c_0 \in rep,\ f_d^l = \mathsf{fold\text{-}left}\ f\ i\ l\}$$
$$E^{f,l,i}(m,\ rep,\ cre)\ (\text{when } m > 0) = \{$$
$$(\mathsf{if}\ null\ l_d\ \mathsf{then}\ \overline{c_m}\langle v_d^l\rangle\ \mathsf{else}\ \nu c.\overline{fl}\langle f, v_d^l, cdr\ l_d, c\rangle.c(o).\overline{c_m}\langle o\rangle,$$
$$\mathsf{if}\ null\ l_d\ \mathsf{then}\ \overline{c_m}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{fr}\langle f, cdr\ l_d, i, c\rangle.c(o).\overline{c_m}\langle f\ v_d^r\ o\rangle),$$
$$(\nu c.\overline{fl}\langle f, v_d^l, cdr\ l_d, c\rangle.c(o).\overline{c_m}\langle o\rangle,$$
$$\nu c.\overline{fr}\langle f, cdr\ l_d, i, c\rangle.c(o).\overline{c_m}\langle f\ v_d^r\ o\rangle),$$
$$(\overline{fl}\langle f, v_d^l, cdr\ l_d, c_{m-1}\rangle.c_{m-1}(o).\overline{c_m}\langle o\rangle,$$
$$\overline{fr}\langle f, cdr\ l_d, i, c_{m-1}\rangle.c_{m-1}(o).\overline{c_m}\langle f\ v_d^r\ o\rangle),$$
$$(c_{m-1}(o).\overline{c_m}\langle o\rangle,\ c_{m-1}(o).\overline{c_m}\langle f\ v_d^r\ o\rangle),$$
$$(\overline{c_m}\langle f_d^l\rangle,\ \overline{c_m}\langle f_d^r\rangle),\quad (P,\ Q)\ \ \big|$$
$$c_m \in rep,\ c_{m-1} \in rep',\ rep' \cap cre = \emptyset,\ rep' \text{ finite},$$
$$d = (\mathsf{length}\ l) - m,\ l_d = \mathsf{drop}\ d\ l,$$
$$v_d^l = \mathsf{fold\text{-}left}\ f\ i\ (\mathsf{take}\ d\ l),\ v_d^r = \mathsf{nth}\ d\ l,$$
$$f_d^l = \mathsf{fold\text{-}left}\ f\ i\ l,\ f_d^r = \mathsf{fold\text{-}right}\ i\ f\ l_d,$$
$$(P,\ Q) \in E^{f,l,i}(m-1,\ rep',\ rep' \cup cre)\}$$

Fig. 2. The partitions of the bisimulation $\mathcal{X}$

preceding sections, we remind our reader that first-order names and constants are easily added to the theory of environmental bisimulation [19]).

We then want to prove equivalent the configurations $a, b, fl \vdash P$ and $a, b, fr \vdash Q$ with public a and b, and where:

$$P = b(f,l,i,k).(\overline{fl}\langle f,i,l,k\rangle \mid a[L])$$
$$Q = b(f,l,i,k).(\overline{fr}\langle f,l,i,k\rangle \mid a[R])$$

To prove their equivalence, we provide a (strong) bisimulation $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$ as in Fig. 2 (where we use sans-serif fonts to denote Haskell-like functions). We will henceforth use the acronyms LHS and RHS for respectively the left-hand and right-hand sides of the bisimulation, i.e. the tested configurations. In this particular example, the same pairs of transitions verify the bisimulation clauses on both LHS and RHS's transitions; we will therefore only consider the transitions of LHS to avoid redundancy. We now analyse $\mathcal{X}_1$, which contains the configurations we want to identify. First, we observe that the set r in $\mathcal{X}_1$ contains at least the public names of P and Q, as required clause 5 of the bisimulation. Also, since the environment is empty, clause 4 is vacuously satisfied. Then, we consider the transitions, starting with $(fl, r \vdash P) \xrightarrow{b(f,i,l,k)} (fl, r \vdash \overline{fl}\langle f,i,l,k\rangle \mid a[L])$, which is matched by $(fr, r \vdash Q) \xrightarrow{b(f,i,l,k)} (fr, r \vdash \overline{fr}\langle f,l,i,k\rangle \mid a[R])$ so that membership to $\mathcal{X}_2$ is satisfied (by taking $n = 1$ and $a_1 = a$, with up-to environment since $\emptyset \subseteq \{(`L, `R)\}$). The name k was added to r in $\mathcal{X}_1$ by clause 5 of the bisimulation, and then input by clause 2.

Then, the elements of $\mathcal{X}_2$ must verify the spawn clause since their environment $\{(`L, `R)\}$ is not empty; spawning $l[L]$ and $l[R]$ for some $l \in r$ just enlarges the products $\prod_{j=1}^{n} a_j[\,]$ by one element, preserving the membership to $\mathcal{X}_2$. Conversely, they can also do a passivation transition (which is a form of higher-order output): a pair $(`L, `R)$ is necessarily added to the environment (to which it already belongs) and the products shrink by one element; membership to $\mathcal{X}_2$ is thus preserved again. Finally, the reaction of $\overline{fl}$ with L (resp. $\overline{fr}$ with R) gives $(\{(`L, `R)\},\ r,\ r\,fl,\ \prod_{j=1}^{n-1} a_j[L] \mid a_n[L \mid P_0],\ r\,fr,\ \prod_{j=1}^{n-1} a_j[R] \mid a_n[R \mid Q_0])$ with $P_0 = \mathsf{if}\ null\ l\ \mathsf{then}\ \overline{k}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{fl}\langle f, f\ i\ (car\ l), cdr\ l, c\rangle.c(m).\overline{k}\langle m\rangle$ and $Q_0 = \mathsf{if}\ null\ l\ \mathsf{then}\ \overline{k}\langle i\rangle\ \mathsf{else}\ \nu c.\overline{fr}\langle f, cdr\ l, i, c\rangle.c(m).\overline{k}\langle f\ (car\ l)\ m\rangle$, which belongs to $\mathcal{X}_3$ up-to environment since $\{(`L, `R)\} \subseteq \mathcal{E}$.

We now show that $\mathcal{X}_3$ satisfies the clauses of environmental bisimulation. Because all locations a_j in $\mathcal{X}_3$ are public and may thus lead to passivation, all P_j, Q_j must be in the environment (by clause 3 of the bisimulation); conversely, all terms from the environment must be spawnable an arbitrary number of times as located processes (by clause 4). It is straightforward to verify that $\mathcal{X}_3$ satisfies these constraints by definition. Respect of clause 5 is also immediate to verify.

We then remark that $\mathcal{X}_3$ contains only locations a_j hosting elements of $\mathcal{E}$, i.e. the fold servers L and R in parallel with their related continuations A_h and B_h (if any). Therefore, in order to analyse the other transitions of elements of $\mathcal{X}_3$, we morally just have to consider the transitions of the elements of $\mathcal{E}$, i.e. the servers and their continuations. Those continuations are members of the set $E^{f,l,i}(\mathsf{length}\ l,\ \{k\},\ r\,fl\,fr)$, where E is a recursive function parametric in several values (see Fig. 2). Concretely, the fixed parameters are the function f to fold, the initial list l and the initial value i. The varying parameters are the number m of elements yet to fold (hence $d = (\mathsf{length}\ l) - m$ is the current "depth" in the whole fold), a set rep of channels to return the result of the current recursive call, and a set cre of already created names.

Let us therefore consider first the transitions of $P_d = \mathsf{if}\ null\ l_d\ \mathsf{then}\ \overline{c_m}\langle v_d^l\rangle\ \mathsf{else}\ \ldots$ and related $Q_d = \mathsf{if}\ null\ l_d\ \mathsf{then}\ \overline{c_m}\langle i\rangle\ \mathsf{else}\ \ldots$ in some $E_m = E^{f,l,i}(m,\ rep,\ cre)$. If this is the "last recursive call," i.e. $m = 0$, then $l_d = []$ and $P_d \xrightarrow{\tau} \overline{c_m}\langle f_d^l\rangle$ where $f_d^l = \mathsf{fold\text{-}left}\ f\ i\ l$ is the final value (by definition of left fold), and $Q_d \xrightarrow{\tau} \overline{c_m}\langle i\rangle$ (by definition too). Since $\overline{c_m}\langle f_d^l\rangle$ and $\overline{c_m}\langle i\rangle$ are also related by E_m, these transitions preserve membership to $\mathcal{X}_3$.

If $m > 0$ (i.e. the recursive call is not the last one), then the else branches are taken in both processes, giving $P'_d = \nu c.\overline{fl}\langle f, v_d^l, cdr\ l_d, c\rangle.c(o).\overline{c_m}\langle o\rangle$ in LHS and $Q'_d = \nu c.\overline{fr}\langle f, cdr\ l_d, i, c\rangle.c(o).\overline{c_m}\langle f\ v_d^r\ o\rangle)$ in RHS, still preserving membership to $\mathcal{X}_3$ since $(P'_d, Q'_d) \in E_m$.

Then, P'_d and Q'_d can both create a name c_{m-1} to become $P''_d = \overline{fl}\langle f, v_d^l, cdr\ l_d, c_{m-1}\rangle.c_{m-1}(o).\overline{c_m}\langle o\rangle$ and $Q''_d = \overline{fr}\langle f, cdr\ l_d, i, c_{m-1}\rangle.c_{m-1}(o).\overline{c_m}\langle f\ v_d^r\ o\rangle)$, provided c_{m-1} has not already been created, i.e. that c_{m-1} is not in cre. But remember that, by definition of $\mathcal{X}_3$ (that verifies clause 4 of

the bisimulation), continuations P'_d and Q'_d can be spawned several times since they belong to $\mathcal{E}$ (along with L and R), and that each copy can thus create its own name c_{m-1}. Therefore, we must relate several (P''_d, Q''_d) all with their own fresh c_{m-1}; the set rep' exactly contains every such c_{m-1}. Notice that the names of rep' are free, allowing the definition of $\mathcal{X}_3$ to list them as $\{\widetilde{c}\}$, the set of names created by the folds.

Now, in order for P''_d to do a transition on private name fl, it must react with a server L, modelling a recursive call to the left fold on the rest of the current list. In this case, not only does P''_d reduce to $P'''_d = c_{m-1}(o).\overline{c_m}\langle o\rangle$, but the replication drawn from L turns into $P_{d+1} =$ if $null$ l_{d+1} then $\overline{c_{m-1}}\langle v^l_{d+1}\rangle$ else .... Naturally, Q''_d follows as well, giving $Q'''_d = c_{m-1}(o).\overline{c_m}\langle f\ v^r_d\ o\rangle$ and $Q_{d+1} =$ if $null$ l_{d+1} then $\overline{c_{m-1}}\langle v^r_{d+1}\rangle$ else .... Since (P'''_d, Q'''_d) belongs to E_m and (P_{d+1}, Q_{d+1}) as well (since $E_m \supseteq E_{m-1} = E^{f,l,i}(m-1,\ rep',\ rep' \cup cre)$ by definition), membership to $\mathcal{X}_3$ is still preserved. Notice that *any* L (and related R) may be used for the above reaction, even one at a location where some other continuations already exist. Because the P_{d+1} and Q_{d+1} add up next to the server they come from, the definition of $\mathcal{E}$ in $\mathcal{X}_3$ contains products of arbitrary length $\prod_h A_h$ and $\prod_h B_h$ in parallel with L and R. Analysis of the transitions of P_{d+1} and Q_{d+1} is the same as that of the transitions of P_d and Q_d and needs no further development.

Then, in order for P'''_d to do a transition, it must react on $c_{m-1} \in rep'$. By definition, the only processes that can send on c_{m-1} are $\overline{c_{m-1}}\langle f^l_{d+1}\rangle$ and $\overline{c_{m-1}}\langle f^r_{d+1}\rangle$ in $E_{m-1} \subseteq E_m$. Then P'''_d reacts with $\overline{c_{m-1}}\langle f^l_{d+1}\rangle$ and turns into $\overline{c_m}\langle f^l_d\rangle$ (with $f^l_d = f^l_{d+1} =$ fold-left $f\ i\ l$). Similarly, the process $\overline{c_{m-1}}\langle f^r_{d+1}\rangle$ reacts with Q'''_d which then turns into $\overline{c_m}\langle f\ v^r_d\ f^r_{d+1}\rangle$, i.e. $\overline{c_m}\langle f^r_d\rangle$, again preserving membership to $\mathcal{X}_3$.

Finally, the processes $\overline{c_m}\langle f^l_d\rangle$ and $\overline{c_m}\langle f^r_d\rangle$ may behave differently depending on where c_m comes from. If c_m is private, then they can react with some continuations $c_m(o).\overline{c_{m+1}}\langle o\rangle$ and $c_m(o).\overline{c_{m+1}}\langle f\ v_{r_{d-1}}\ o\rangle$) that are related by $E_{m+1} = E^{f,l,i}(m+1,\ rep'',\ cre \setminus rep)$. Then, $\overline{c_m}\langle f^l_d\rangle$ and $\overline{c_m}\langle f^r_d\rangle$ both turn into 0 while the continuations' continuations are still related by E_{m+1}, like we showed for E_m in the previous paragraph. (There are no other reactions between elements of E_m and E_n with $m \neq n$.) Otherwise, necessarily $c_m = k$ by definition of $\mathcal{X}_3$ and thus, by definition of E, the same value fold-left $f\ i\ l =$ fold-right $f\ l\ i$ is output to public channel k.

This concludes our proof that elements of $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$ satisfy the clauses of environmental bisimulations (up-to environment), and thus that $a, b, fl \vdash P$ and $a, b, fr \vdash Q$ are bisimilar with public names a and b.

VI. Non-Bisimilarity and Similarity

A. Processes with Different Number of Locations Used

In Section V, we compared processes that recurse the same number of times and built bisimulations relating these processes such that whenever a process uses a location so does the other. We illustrate now that, in our distributed setting (regardless of the semantics of ν), the number of locations used *does* matter to draw some bisimilarity results. We then show how (mutual) simulation gets round this limitation.

$P = !pow_{lin}(a,b,k).$if $b = 0$
then $\overline{k}\langle 1\rangle$ else $\nu c.\overline{pow_{lin}}\langle a, b{-}1, c\rangle.c(m).\overline{k}\langle a{\times}m\rangle$
$Q = !pow_{log}(a,b,k).$if $b = 0$
then $\overline{k}\langle 1\rangle$ else if $b\%2 = 0$ then $\overline{pow_{log}}\langle a{\times}a, b{\div}2, k\rangle$
else $\nu c.\overline{pow_{log}}\langle a, b{-}1, c\rangle.c(m).\overline{k}\langle a{\times}m\rangle$

Fig. 3. Linear and logarithmic power functions

Let us consider two implementations of the power function $pow(a,b) = a^b$, one of linear complexity, and the other of logarithmic complexity, as shown in Fig. 3. Suppose that we want to replace the linear implementation P by the logarithmic one Q in a distributed system, and to check if there will be no visible difference. We could model those systems as $l[P] \mid \overline{pow_{lin}}\langle a, b, k\rangle$ and $l[Q] \mid \overline{pow_{log}}\langle a, b, k\rangle$ for integers a, b and public names l, k, and show their equivalence using our proof technique, trying to build a bisimulation $\mathcal{X}$ relating them, starting with $(\emptyset, lk, lk\, pow_{lin}, l[P] \mid \overline{pow_{lin}}\langle a, b, k\rangle, lk\, pow_{log}, l[Q] \mid \overline{pow_{log}}\langle a, b, k\rangle) \in \mathcal{X}$. We consider the situation where location l is passivated and then spawned b times: $(\emptyset, lk, lk\, pow_{lin}, \prod_{i=1}^{b} l_i[P] \mid \overline{pow_{lin}}\langle a, b, k\rangle, lk\, pow_{log}, \prod_{i=1}^{b} l_i[Q] \mid \overline{pow_{log}}\langle a, b, k\rangle) \in \mathcal{X}$. Let us consider now the state where the linear version has unfolded all the b recursive calls across $l_1, \ldots, l_b$, like: $l_1[P \mid c_1(x).\overline{k}\langle a{\times}x\rangle] \mid l_2[P \mid c_2(x).\overline{c_1}\langle a{\times}x\rangle] \mid \ldots \mid l_b[P \mid \overline{c_{b-1}}\langle 1\rangle]$. Similarly, the logarithmic version should (somehow) follow weakly: $l_1[Q \mid \ldots] \mid l_2[Q \mid \ldots] \mid \ \ldots \mid l_b[Q \mid \ldots]$. Suppose now that the attacker passivates the location l_1 that contains $c_1(x).\overline{k}\langle a{\times}x\rangle$, so that the left hand-side of the bisimulation can no longer return its result. Then, by the definition of bisimilarity, the logarithmic implementation too must not return a value if l_1 is passivated. The attacker may now spawn back the passivated contents of l_1, and then repeat the same passivation test on each of l_2, $l_3 \ldots$. In the end, we know that more than $log(b)$ locations were necessary for the recursive calls of the logarithmic version of the power function, which is impossible by design since this implementation can do at most $log(b)$ recursive calls. The two systems thus cannot be bisimilar.

Although bisimilarity does not hold between the distributed linear power function and the logarithmic one, we prove that mutual simulation does, by crafting simulations relating them. We first build the simulation $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_2$ as in Fig. 4, in a manner very similar to that of Section V. Our explanations below will thus focus on main differences from Section V. First, we recall that, because we consider (weak) simulations, we require that transitions by LHS be (weakly) matched by transitions on RHS, but not the converse. Thus, in this example, while the simulation $\mathcal{Y}$ needs to keep in LHS all the intermediate states of the linear power calculation, it suffices to keep in RHS only the initial and final states of the logarithmic power calculation.

The initial states are related by $\mathcal{Y}_1$. As far as intermediate states are concerned, we decide that the processes of LHS that can do an observable action (i.e. an output to k) be related to

$$\mathcal{Y}_1 = \{(\mathcal{E}, r, r\ pow_{lin}, \prod_i^n l_i[P] \mid \overline{pow_{lin}}\langle a,b,k\rangle,$$
$$r\ pow_{log}, \prod_i^n l_i[Q] \mid \overline{pow_{log}}\langle a,b,k\rangle) \mid$$
$$k, \widetilde{l} \in r,\ \mathcal{E} \subseteq \{(\text{`}P, \text{`}Q)\}\}$$
$$\mathcal{Y}_2 = \{(\mathcal{E}, r, \widetilde{c}\, r\ pow_{lin}, \prod_i l_i[P_i], \widetilde{d}\, r\ pow_{log}, \prod_i l_i[Q_i]) \mid$$
$$\mathcal{E} = \{(P \mid \prod_h A_h,\ Q \mid \prod_h B_h) \mid$$
$$(\widetilde{A}, \widetilde{B}) \in E^{a,b}(b, \{k\}, pow_{lin}\ pow_{log}\ r)\},$$
$$(\text{`}\widetilde{P}, \text{`}\widetilde{Q}) \in \mathcal{E},\ k, \widetilde{l} \in r,\ \widetilde{c} = fn(\mathcal{E}.1) \setminus r \setminus \{pow_{lin}\},$$
$$\widetilde{d} \cap r \setminus \{pow_{log}\} = \emptyset,\ |\widetilde{d}| = depth(b),$$
$$depth(x) = \begin{cases} 0 & \text{if } x = 0 \\ depth(x/2) & \text{if } x\%2 = 0 \\ 1{+}depth(x{-}1) & \text{otherwise}\} \end{cases}$$
$$E^{a,b}(0, rep, cre) = \{$$
$$(\text{if } 0 = 0 \text{ then } \overline{c_0}\langle 1\rangle \text{ else } \ldots,\ R),\ (\overline{c_0}\langle 1\rangle,\ R) \mid$$
$$c_0 \in rep,\ R = \overline{c_0}\langle 1\rangle \text{ if } b = 0, \text{otherwise } R = 0\}$$
$$E^{a,b}(m, rep, cre) \text{ (when } m > 0) = \{$$
$$(\text{if } m = 0 \text{ then } \ldots \text{ else } \nu c.\overline{pow_{lin}}\langle a, m{-}1, c\rangle.c(o).\overline{c_m}\langle a{\times}o\rangle,\ R),$$
$$(\nu c.\overline{pow_{lin}}\langle a, m{-}1, c\rangle.c(o).\overline{c_m}\langle a{\times}o\rangle,\ R),$$
$$(\overline{pow_{lin}}\langle a, m{-}1, c_{m-1}\rangle.c_{m-1}(o).\overline{c_m}\langle a{\times}o\rangle,\ R),$$
$$(c_{m-1}(o).\overline{c_m}\langle a{\times}o\rangle,\ R),\ (\overline{c_m}\langle a^m\rangle,\ R),\ (P_{m-1},\ 0) \mid$$
$$c_m \in rep,\ c_{m-1} \in rep',\ rep' \cap cre = \emptyset,\ rep' \text{ finite},$$
$$P_{m-1} \in E^{a,b}(m{-}1, rep', rep' \cup cre),$$
$$R = \overline{c_m}\langle a^b\rangle \text{ if } m = b, \text{otherwise } R = 0\}$$

Fig. 4. Simulation $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_2$ between linear and logarithmic power functions

processes in RHS able to do the same action, so as to guarantee satisfaction of the output clause of simulation. Therefore, we define $\mathcal{Y}_2$ such that subprocesses of LHS that have k free in them (i.e. continuations of the initial call to pow_{lin}) are related to $\overline{k}\langle a^b\rangle$ on RHS (i.e. final state of the call to pow_{log}), and that other subprocesses of LHS are related to 0 on RHS.

We now show that the set $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_2$ is a weak environmental simulation. We start with $\mathcal{Y}_1$ that relates (for $n = 1$ and $l_1 = l$) the processes we want to prove related: $lk\ pow_{lin} \vdash l[P] \mid \overline{pow_{lin}}\langle a,b,k\rangle$ and $lk\ pow_{log} \vdash l[Q] \mid \overline{pow_{log}}\ \langle a,b,k\rangle)$. Suppose that the client $\overline{pow_{lin}}\langle a,b,k\rangle$ of LHS reacts with located server $l_i[P]$, leaving a process $P_b = (\text{if } b = 0 \text{ then } \overline{k}\langle 1\rangle \text{ else } \nu c.\overline{pow_{lin}}\langle a, b{-}1, c\rangle.c(m).\overline{k}\langle a{\times}m\rangle)$ at l_i. RHS can follow by reacting weakly (doing all calculations on the spot) with $l[Q]$, leaving process $l_i[\overline{k}\langle a^b\rangle]$. The rests are related by $\mathcal{Y}_2$ up-to environment since $\{(\text{`}P, \text{`}Q),\ (\text{`}P | P_b, \text{`}Q | \overline{k}\ \langle a^b\rangle)\} \subseteq \mathcal{E}$.

Then, if $b = 0$, P_b reduces to $\overline{k}\langle 1\rangle$, while RHS's $\overline{k}\ \langle a^b\rangle = \overline{k}\langle 1\rangle$ follows weakly by not doing any transition. The continuations $\overline{k}\langle 1\rangle$ and $\overline{k}\langle 1\rangle$ preserve membership to $\mathcal{Y}_2$. Moreover, if LHS's $\overline{k}\langle 1\rangle$ outputs 1 to k, then so can RHS's $\overline{k}\ \langle 1\rangle$ as expected.

Otherwise, if $b > 0$, P_b can reduce successively to $P'_b = \nu c.\ \overline{pow_{lin}}\langle a, b{-}1, c\rangle.\ldots$ and to $P''_b = \overline{pow_{lin}}\langle a, b{-}1, c_b\rangle.\ldots$. In RHS, $\overline{k}\langle a^b\rangle$ follows both transitions weakly, still preserving membership to $\mathcal{Y}_2$.

Then P''_b can react with a server P: P''_b becomes $P'''_b = c_{b-1}(o).\overline{k}\langle a{\times}o\rangle$, P becomes $P \mid P_{b-1} = P \mid \text{if } (b{-}1) = 0 \text{ then } \overline{c_{b-1}}\langle 1\rangle \text{ else } \nu c.\overline{pow_{lin}}\langle a, b{-}2, c\rangle.c(m).\overline{c_{b-1}}\langle a{\times}m\rangle$, and $\overline{k}\langle a^b\rangle$ follows weakly. For example, we have $l_1[P \mid P''_b] \mid l_2[P] \xrightarrow{\tau} l_1[P \mid P'''_b] \mid l_2[P \mid P_{b-1}]$ in LHS, and $l_1[Q \mid \overline{k}\ \langle a^b\rangle] \mid l_2[Q] \xrightarrow{\tau} l_1[Q \mid \overline{k}\langle a^b\rangle] \mid l_2[Q \mid 0]$ in RHS. It results from the above that P'''_b is related to $\overline{k}\langle a^b\rangle$, and P_{b-1} to 0 since no process was added in RHS by the weak transition. Membership to $\mathcal{Y}_2$ is still satisfied.

We skip the analysis of transitions of P_{b-1}, as it similar to that of transitions of P_b.

If P'''_b inputs on c_{b-1}, it becomes $P''''_b = \overline{k}\langle a^b\rangle$ because the only output to c_{b-1} comes from $\overline{c_{b-1}}\langle a^{b-1}\rangle$ in $E^{a,b}(b{-}1, rep', rep' \cup cre)$. RHS follows weakly, still giving $\overline{k}\langle a^b\rangle$, and preserving membership to $\mathcal{Y}_2$. Finally, output of a^b to k by both processes can happen, satisfying the simulation's output clause.

This concludes our proof that elements of $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_2$ satisfy the clauses of environmental simulation (up-to environment), and thus that the distributed linear algorithm approximates the logarithmic one.

With the same approach, we may easily show that the linear algorithm simulates the logarithmic one as well (the previous simulation proof does not depend on the number of reduction steps on the left-hand side). This means that the two algorithms simulate each other even though they are not bisimilar, supporting the usefulness of mutual simulation in higher-order distribution.

B. Processes with Different Number of Names Created

In the introduction, we gave an example of perhaps surprising (but rational) non-bisimilarity between located processes $l[\nu a.\nu b.P]$ and $l[\nu b.\nu a.P]$. A possibly even more surprising example would be the following:

$$\{l_1, \omega\} \vdash l_1[\nu a.(a|\overline{a}.\overline{\omega})] \not\approx_{\{\omega, l_1\}} \{l_1, \omega\} \vdash l_1[\nu a.\nu b.(a.b|\overline{a}.\overline{b}.\overline{\omega})]$$

To see why these configurations are not bisimilar, we consider the duplication of the located processes after the name creations; for $s = \{l_1, l_2, a, \omega\}$, we have:

$$s \vdash l_1[a \mid \overline{a}.\overline{\omega}] \mid l_2[a \mid \overline{a}.\overline{\omega}] \not\approx_{\{\omega, l_1, l_2\}}$$
$$s, b \vdash l_1[a.b \mid \overline{a}.\overline{b}.\overline{\omega}] \mid l_2[a.b \mid \overline{a}.\overline{b}.\overline{\omega}]$$

Consider now the transition of the right-hand side:

$$s \vdash l_1[a.b \mid \overline{a}.\overline{b}.\overline{\omega}] \mid l_2[a.b \mid \overline{a}.\overline{b}.\overline{\omega}] \xrightarrow{\tau} s \vdash l_1[b \mid \overline{a}.\overline{b}.\overline{\omega}] \mid l_2[a.b \mid \overline{b}.\overline{\omega}]$$

To match, the left-hand side may do a weak transition to one of the six following processes:

$$l_1[\overline{\omega}] \mid l_2[\overline{\omega}] \qquad l_1[a \mid \overline{a}.\overline{\omega}] \mid l_2[\overline{\omega}] \qquad l_1[\overline{\omega}] \mid l_2[a \mid \overline{a}.\overline{\omega}]$$
$$l_1[a \mid \overline{\omega}] \mid l_2[\overline{a}.\overline{\omega}] \qquad l_1[\overline{a}.\overline{\omega}] \mid l_2[a \mid \overline{\omega}] \qquad l_1[a \mid \overline{a}.\overline{\omega}] \mid l_2[a \mid \overline{a}.\overline{\omega}]$$

with created names s. Suppose that the attacker then passivates l_1 on the right-hand side:

$$s \vdash l_1[b \mid \overline{a}.\overline{b}.\overline{\omega}] \mid l_2[a.b \mid \overline{b}.\overline{\omega}] \xrightarrow{\overline{l_1}\langle \text{`}b|\overline{a}.\overline{b}.\overline{\omega}\rangle} s \vdash l_2[a.b \mid \overline{b}.\overline{\omega}]$$

The resulting process is stuck, so because of the symmetry of bisimulations, the left-hand side must be able to passivate l_1 and become stuck too. The only way to achieve this is necessarily by doing a transition $\xrightarrow{\overline{l_1}\langle \text{`}a|\overline{\omega}\rangle} l_1, l_2, a, \omega \vdash l_2[\overline{a}.\overline{\omega}]$. The attacker can then passivate the contents of l_2 in both sides of the bisimulation, and be left with processes 0. Now, he can spawn back what was output during the passivation of l_1, and we thus have:

$$s \vdash l_1[a \mid \overline{\omega}] \not\approx_{\{\omega, l_1, l_2\}} s, b \vdash l_1[b \mid \overline{a}.\overline{b}.\overline{\omega}]$$

Obviously, the right-hand side is stuck, but the left one can exhibit $\overline{\omega}$, thus proving that the two processes are not bisimilar.

By the same reasoning as in Section VI-A, the above processes as well as the non-bisimilar ones in the introduction can also be shown to be mutually similar.

VII. Discussions

We defined HOπPn, the higher-order distributed π-calculus with passivation and name creation, and developed its equivalence and inequivalence theories. Although many of the inequivalences may have been counter-intuitive, we emphasise that they are rational in hindsight and reflect the reality of non-linear higher-order distribution (not necessarily passivation but also duplication in general; cf. [4]). One may suggest (environmental) ready simulations [2] for defining a weaker equivalence than bisimilarity, but their branching-time nature also distinguishes all the non-bisimilar examples of this paper. A linear-time equivalence like testing equivalence [13] may deserve consideration.

Recently studied higher-order distributed process calculi include the Kell calculus [20], Homer [8] and the higher-order π-calculus with passivation (HOπP) [11]. They are extensions of the π-calculus with the communication of processes and their execution inside locations, and all have name restriction semantics. (Other distributed process calculi such as ambients [3] and Dpi [7] identify name creation and name restriction semantics, but are not higher-order in our sense of passing of processes through channels.) Research on the Kell calculus and Homer led to defining sound and complete *context bisimulations* [16]. However, they critically rely on universal quantification on contexts and are almost as hard as reduction-closed barbed equivalence as proof methods. Later, Lenglet et al. [11] focused on HOπP, a calculus simpler than both the Kell calculus and Homer. In addition to sound and complete context bisimulations, they provided more practical *normal bisimulations* [16] that are sound and complete in the *absence* of name restriction but are unsound otherwise. Also for HOπP, Piérard and Sumii defined a sound but incomplete environmental bisimulation proof technique [15] with strong constraints (the environments could not contain any restriction operator nor higher-order inputs). Even though non-trivial equivalences of processes which could not be realistically proven with context bisimulations can be proven with this technique, the constraints have a severe impact on the variety of processes that can be considered.

The simplicity of HOπPn, notably the transparency of locations, does not offer enough control over the communications between processes, and therefore hinders natural modelling of real systems where processes cannot freely interact with one another. Such systems can be modelled with non-transparent locations [20], [8], e.g. locations that only allow communications between processes from the same level or one level above/below. Moreover, passivation in HOπPn unifies failure, migration, and duplication as higher-order outputs, therefore mixing different behaviours. Even though identifying them keeps the model simple, their distinction may enable a more realistic modelling of higher-order distributed systems.

Environmental bisimulations have also been proposed [1] for λ-calculus with name creation [21]. In the first-order π-calculus, name creation and name restriction are equivalent [10]; we conjecture this also holds in the higher-order π-calculus. Fernández and Gabbay [5] extended first-order term rewriting with the $\mathcal{N}$ operator [6]. Their semantics is similar to name restriction rather than creation.

References

[1] N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus. http://www.scss.tcd.ie/Vasileios.Koutavas/publications/nu-calc-submitted.pdf, unpublished, 2010.

[2] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[3] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[4] G. Castagna, J. Vitek, and F. Z. Nardelli. The Seal Calculus. *Information and Computation*, 201(1):1–54, 2005.

[5] M. Fernández and M. J. Gabbay. Nominal rewriting with name generation: abstraction vs. locality. In *Principles and Practice of Declarative Programming*, pages 47–58. ACM, 2005.

[6] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[7] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 1998.

[8] T. Hildebrandt, J. C. Godskesen, and M. Bundgaard. Bisimulation congruences for Homer: a calculus of higher-order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen, 2004.

[9] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[10] Y. Kawabe, K. Mano, E. Horita, and K. Kogure. Name creation implements restriction in the π-calculus. *Systems and Computers in Japan*, 36:78–91, 2005.

[11] S. Lenglet, A. Schmitt, and J.-B. Stefani. Normal bisimulations in calculi with passivation. In *Foundations of Software Science and Computational Structures*, volume 5504 of *LNCS*, pages 257–271. Springer, 2009.

[12] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[13] R. D. Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[14] A. Piérard and E. Sumii. Appendix to a higher-order distributed calculus with name creation. http://www.kb.ecei.tohoku.ac.jp/~adrien/pubs/AppendixCreation.pdf.

[15] A. Piérard and E. Sumii. Sound bisimulations for higher-order distributed process calculus. In *Foundations of Software Science and Computational Structures*, volume 6604 of *LNCS*, pages 123–137. Springer, 2011.

[16] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

[17] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33:5:1–5:69, 2011.

[18] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[19] N. Sato and E. Sumii. The higher-order, call-by-value applied pi-calculus. In *Asian Symposium on Programming Languages and Systems*, volume 5904 of *LNCS*, pages 311–326. Springer, 2009.

[20] A. Schmitt and J.-B. Stefani. The Kell calculus: A family of higher-order distributed process calculi. In *Global Computing*, volume 3267 of *LNCS*, pages 146–178. Springer, 2004.

[21] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.

[22] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1-3):169–192, 2007.

[23] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54:1–43, 2007.