Open Multiparty Interaction*

Chiara Bodei[1], Linda Brodo[2], and Roberto Bruni[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Dipartimento di Scienze Politiche, Scienze della Comunicazione e Ingegneria dell'Informazione, Università di Sassari, Italy

Abstract. We present the `link`-calculus, a process calculus based on interactions that are *multiparty*, i.e., that may involve more than two processes and are *open*, i.e., the number of involved processes is not fixed or known a priori. Communications are seen as chains of links, that record the source and the target ends of each hop of interactions. The semantics of our calculus mildly extends the one of CCS in the version without message passing, and the one of π-calculus in the full version. Cardelli and Gordon's Mobile Ambients, whose movement interactions we show to be inherently open multi-party, is encoded in our calculus in a natural way, thus providing an illustrative example of its expressiveness.

Introduction

An *interaction* is an action by which communicating processes can influence each other. Interactions in the time of the Web are something more than input and output between two entities. Actually, the word itself can be misleading, by suggesting a reciprocal or mutual kind of actions. Instead, interactions more and more often involve many parties and actions are difficult to classify under output and input primitives. We can imagine an interaction as a sort of puzzle in which many pieces have to be suitably combined together in order to work.

As networks have become part of the critical infrastructure of our daily activities (for business, social, health, government, etc.) and a large variety of loosely coupled processes have been offered over global networks, as services, more sophisticated forms of interactions have emerged, for which convenient formal abstractions are under study. For example, one important trend in networking is moving towards architectures where the infrastructure itself can be manipulated by the software, like in the Software Defined Networking approach, where the control plane is remotely accessible and modifiable by software clients, using open protocols such as OpenFlow, making it possible to decouple the network control from the network topology and to provide Infrastructure as a Service over data-centers and cloud systems. Another example is that of complex biological interactions as the ones emerging in bio-computing and membrane systems.

As a consequence, from a foundational point of view, it is strategic to provide the convenient formal abstractions and models to naturally capture these new

* Research partially supported by the EU through the FP7-ICT Integrated Project 257414 ASCEns (Autonomic Service-Component Ensembles).

N. Martí-Oliet and M. Palomino (Eds.): WADT 2012, LNCS 7841, pp. 1–23, 2013.

communication patterns, by going beyond the ordinary binary form of communication. These models should be sufficiently expressive to faithfully describe the complex phenomena, but they have also to provide a basis for the formal analysis of such systems, by offering sufficient mathematical structure and tractability. We present here a process calculus, called `link`-calculus, which takes interaction as its basic ingredient. The described interactions are *multiparty*, i.e., they may involve more than two processes and are *open*, i.e., the number of involved processes is not known *a priori*. Communication actions are given in terms of links, that record the source and the target ends of each hop of interactions. Links can be indeed combined in link chains that route information across processes from a source to a destination. Despite the inherent complexity of representing more sophisticated forms of interaction, we show that the underlying synchronisation algebra and name handling primitives are quite simple and a straight generalisation of dyadic ones. This is witnessed by the operational semantics rules of our calculus, that in the simpler version (i.e., without message passing) resemble the rules of CCS [21], while in the full one they resemble the rules of π-calculus [22].

Finally, we address a more technical issue, by providing a natural encoding of Cardelli and Gordon's (pure) Mobile Ambients (MA) [9] in the `link`-calculus. We have chosen Mobile Ambients as one of the most representative examples of process calculi for compartmentalisation based on the principles of location mobility and location awareness, from which many other models originated as abstractions for global computing, for systems biology and membrane systems. Our encoding highlights the multi-party nature of interactions in MA and shows that the spatial aspects due to the ambient nesting can be dealt with in passing to a flat calculus such as our `link`-calculus. We prove a tight correspondence at the level of reduction semantics and we provide a new bisimilarity semantics for MA as a side result. We are confident that analogous results can be obtained for many descendants of MA, as e.g., the Brane Calculus [8].

The paper, where we assume the reader has some familiarity with process calculi, is organised as follows. In Sections 1 and 2 we define the `link`-calculus, starting from introducing its fragment without name mobility, called *Core Network Algebra*. In Section 3, we recall the basics of MA. In Section 4, we define the encoding from MA to the `link`-calculus. In Section 5, we draw some final remarks, outline some future research avenues and point to some related works.

1 A Core Network Algebra

We start by defining a network-aware extension of CCS, called *Core Network Algebra* (CNA for short), whose communication actions are given in terms of *links*. A link is a pair that record the source and the target ends of a communication, meaning that the input available at the source end can be forwarded to the target one. Links are combined in *link chains* to describe how information can be routed across ends. Link chains also allow seamless realisation of multiparty synchronisations. While in this section we focus on the basic ideas of link interaction, we shall enhance the model with name mobility in the next section.

Links and Link Chains. Let $\mathcal{C}$ be the set of channels, ranged over by $a, b, c, ...$ Let $\mathcal{C} \cup \{\tau\} \cup \{*\}$ be the set of actions, ranged over by $\alpha, \beta, \gamma, ...$, where the symbol τ denotes a *silent* action, and the symbol $*$ denotes a non-specified action.

A *link* is a pair $\ell = {}^{\alpha}\backslash_{\beta}$; it can be read as forwarding the input available on α to β, and we call α the *source end* of ℓ and β the *target end* of ℓ. A link $\ell = {}^{\alpha}\backslash_{\beta}$ is *valid* if either $\alpha, \beta \neq *$ or $\ell = {}^{*}\backslash_{*}$ In the first case, the link is called *solid*. The link ${}^{*}\backslash_{*}$ is called *virtual*. We let $\mathcal{L}$ be the set of valid links. Examples of non valid links are ${}^{\tau}\backslash_{*}$ and ${}^{*}\backslash_{a}$.

As it will be shortly explained, a virtual link is a sort of "missing link" inside a link chain, a non specified part that can be supplied, as a solid link, by another link chain, via a suitable composition operation.

A *link chain* is a finite sequence $s = \ell_1 ... \ell_n$ of (valid) links $\ell_i = {}^{\alpha_i}\backslash_{\beta_i}$ s.t.:

1. for any $i \in [1, n-1]$, $\begin{cases} \beta_i, \alpha_{i+1} \in \mathcal{C} & \text{implies } \beta_i = \alpha_{i+1} \\ \beta_i = \tau & \text{iff } \alpha_{i+1} = \tau \end{cases}$
2. if $\forall i \in [1, n].\alpha_i, \beta_i \in \{\tau, *\}$, then $\forall i \in [1, n].\alpha_i = \beta_i = \tau$.

The first condition says that any two adjacent solid links must agree on their ends: it also imposes that τ cannot be matched by $*$. The second condition disallows chains made of virtual links only.

The empty link chain is denoted by ϵ. A non-empty link chain is *solid* if all its links are so. A link chain is *simple* if it includes exactly one solid link (and one, none or many virtual links). For ℓ a solid link and s a simple link chain, we write $\ell \in s$ if ℓ is the *only* solid link occurring in s. We write $|s|$ to denote the *length* of s, i.e., the number of links in s.

We say that an action a is *matched* in s if: 1) $\alpha_1, \beta_n \neq a$, and 2) for any $i \in [1, n-1]$, either $\beta_i = \alpha_{i+1} = a$ or $\beta_i, \alpha_{i+1} \neq a$. Otherwise, we say that a is *unmatched* (or *pending*) in s. For instance, a is matched in the sequence ${}^{\tau}\backslash^{a}_{a}\backslash_{\tau}$, while it is not matched in the sequences ${}^{\tau}\backslash^{*}_{a}\backslash_{*}$ and in ${}^{a}\backslash^{a}_{a}\backslash^{a}_{a}\backslash_{a}$.
We can rephrase usual communication primitives of process algebras as links.

- The output action $\overline{a}$ (resp. the input action a) of CCS can be seen as the link ${}^{\tau}\backslash_{a}$ (resp. ${}^{a}\backslash_{\tau}$) and the solid link chain ${}^{\tau}\backslash^{a}_{a}\backslash_{\tau}$ as a CCS-like communication.
- The action a of CSP can be seen as the link ${}^{a}\backslash_{a}$ and the solid link chain ${}^{a}\backslash^{a}_{a}\backslash^{a}_{a}\backslash_{a}$ as a CSP-like communication among three peers over a.

The following basic operations over links and link chains are partial and strict, i.e., they may issue $\perp$ (undefined) and the result is $\perp$ if any argument is $\perp$. To keep the notation short: if one of the sub-expressions in the righthand side (RHS) of any defining equation is undefined, then we assume the result is $\perp$; if none of the conditions in the RHS of any defining equation is met, then the result is $\perp$.

Merge. Two link chains can be merged if they are to some extent "complementary", i.e., if they have the same length, each provides links that are not specified in the other and together they form a (valid) link chain. The virtual links in a chain can be seen as the not yet specified part in the chain, and possibly provided

by another link chain after a merge. If there is a position where both link chains carry solid links, then there is a clash and the merge is not possible (undefined). If the merge would result in a non valid sequence, then the merge is not possible. Formally, for $s = \ell_1 ... \ell_n$ and $s' = \ell'_1 ... \ell'_n$, with $\ell_i = {}^{\alpha_i}\backslash_{\beta_i}$ and $\ell'_i = {}^{\alpha'_i}\backslash_{\beta'_i}$ for any $i \in [1, n]$, we define $s \bullet s'$ by letting:[1]

$$s \bullet s' \triangleq (\ell_1 \bullet \ell'_1)...(\ell_n \bullet \ell'_n) \qquad {}^{\alpha}\backslash_{\beta} \bullet {}^{\alpha'}\backslash_{\beta'} \triangleq {}^{(\alpha \bullet \alpha')}\backslash_{(\beta \bullet \beta')} \qquad \alpha \bullet \beta \triangleq \begin{cases} \alpha & \text{if } \beta = * \\ \beta & \text{if } \alpha = * \end{cases}$$

Roughly, the merge is defined element-wise on the actions of a link chain, by ensuring that whenever two actions are merged, (at least) one of them is $*$ and that the result of the merge is still a link chain. Intuitively, we can imagine that s and s' are two parts of the same puzzle separately assembled, where solid links are the pieces of the puzzle and virtual links are the holes in the puzzle and their merge $s \bullet s'$ puts the two parts together, without piece overlaps.

Example 1. Let $s_1 = {}^{\tau}\backslash^{*}_{a}\backslash^{*}_{*}\backslash_{*}$, $s_2 = {}^{*}\backslash^{a}_{*}\backslash^{*}_{b}\backslash_{*}$, and $s_3 = {}^{*}\backslash^{*}_{*}\backslash^{b}_{*}\backslash_{\tau}$ three link chains of the same length. Then s_1 and s_2 can be merged to obtain $s = s_1 \bullet s_2 = ({}^{\tau}\backslash_{a}{}^{*}\backslash_{*})({}^{*}\backslash_{*}{}^{a}\backslash_{b})({}^{*}\backslash_{*}{}^{*}\backslash_{*}) = ({}^{\tau\bullet *}\backslash_{a\bullet *})({}^{*\bullet a}\backslash_{*\bullet b})({}^{*\bullet *}\backslash_{*\bullet *}) = {}^{\tau}\backslash^{a}_{a}\backslash^{*}_{b}\backslash_{*}$. Similarly, s and s_3 can then be merged to obtain: $s \bullet s_3 = {}^{\tau}\backslash^{a}_{a}\backslash^{b}_{b}\backslash_{\tau}$.

Lemma 1. *(i) If s is solid, then for any s' we have $s \bullet s' = \bot$.*
(ii) The merge of link chains is a commutative and associative operation.
(iii) For any ℓ, ℓ': $\ell \bullet \ell' = {}^{*}\backslash_{*}$ if and only if $\ell = \ell' = {}^{*}\backslash_{*}$.

Restriction. Certain actions of the link chain can be hidden by restricting the channel where they take place. Of course, restriction is possible only if this process does not introduce any unmatched communication. Formally, for $s = \ell_1 ... \ell_n$, with $\ell_i = {}^{\alpha_i}\backslash_{\beta_i}$ with $i \in [1, n]$, we define the restriction operation $(\nu\, a)s$ by letting

$$(\nu\, a)s \triangleq {}^{\alpha_1}\backslash(\nu\, a)({}^{\alpha_2}_{\beta_1})\backslash ... \backslash(\nu\, a)({}^{\alpha_n}_{\beta_{n-1}})\backslash_{\beta_n} \text{ if } \alpha_1, \beta_n \neq a \qquad (\nu\, a)({}^{\alpha}_{\beta}) \triangleq \begin{cases} {}^{\tau}_{\tau} & \text{if } \alpha = \beta = a \\ {}^{\alpha}_{\beta} & \text{if } \alpha, \beta \neq a \end{cases}$$

Lemma 2. *(i) For any a, ℓ: $(\nu\, a)\ell = {}^{*}\backslash_{*}$ if and only if $\ell = {}^{*}\backslash_{*}$.*
(ii) For any a, s, s' such that a does not appear in s: $(\nu\, a)(s \bullet s') = s \bullet (\nu\, a)s'$.
(iii) For any a, b, s: $(\nu\, a)(\nu\, b)s = (\nu\, b)(\nu\, a)s$.

Example 2. Let $s = {}^{\tau}\backslash^{a}_{a}\backslash^{*}_{b}\backslash_{*}$ and $s' = {}^{*}\backslash^{*}_{*}\backslash^{b}_{*}\backslash_{\tau}$. Then, $(\nu\, a)s = {}^{\tau}\backslash(\nu\, a)({}^{a}_{a})\backslash(\nu\, a)({}^{*}_{b})\backslash_{*}$ $= {}^{\tau}\backslash^{\tau}_{\tau}\backslash^{*}_{b}\backslash_{*}$. Similarly, $(\nu\, a)(s \bullet s') = {}^{\tau}\backslash^{\tau}_{\tau}\backslash^{b}_{b}\backslash_{\tau} = ((\nu\, a)s) \bullet s'$.

[1] As anticipated, we remark that in the defining equations for merge it is implicitly understood that: if $\ell_i \bullet \ell'_i = \bot$ for some i, then $s \bullet s' = \bot$; if the sequence $(\ell_1 \bullet \ell'_1)...(\ell_n \bullet \ell'_n)$ is not a link chain, then $s \bullet s' = \bot$; if $\alpha \bullet \alpha' = \bot$ or $\beta \bullet \beta' = \bot$, then ${}^{\alpha}\backslash_{\beta} \bullet {}^{\alpha'}\backslash_{\beta'} = \bot$; if $\alpha, \beta \neq *$, then $\alpha \bullet \beta = \bot$.

Process Syntax. The CNA processes are generated by the following grammar (for brevity, we omit the CCS-like renaming operator $P[\Phi]$ because it will be later subsumed by the syntax in Section 2):

$$P, Q ::= \mathbf{0} \mid X \mid \ell.P \mid P + Q \mid P|Q \mid (\nu\, a)P \mid \mathsf{rec}X.\, P$$

where ℓ is a solid link (i.e., $\ell = {}^{\alpha}\backslash_{\beta}$ with $\alpha, \beta \neq *$).

Roughly, processes are built over a CCS-like syntax (with nil process $\mathbf{0}$, prefix $\ell.P$, sum $P + Q$, parallel $P|Q$, restriction $(\nu\, a)P$ and recursion $\mathsf{rec}X.\, P$, for X a process variable), but where the underlying synchronisation algebra is based on link chains. This is made evident by the operational semantics, presented below.

As usual, $(\nu\, a)P$ binds the occurrences of a in P, the sets of free and of bound names of a process P are defined in the obvious way and denoted, respectively, by $fn(P)$ and $bn(P)$, processes are taken up to alpha-conversion of bound names, and we shall often omit trailing $\mathbf{0}$, e.g., by writing ${}^{a}\backslash_{b}$ instead of ${}^{a}\backslash_{b}.\mathbf{0}$

Operational Semantics. The idea is that communication can be routed across several processes by combining the links they make available to form a link chain. Since the length of the link chain is not fixed a priori, an open multiparty synchronisation is realised.

The operational semantics is defined in terms of a Labelled Transition System (LTS) whose states are CNA processes, whose labels are link chains and whose transitions are generated by the SOS rules in Fig. 1. Notice that the SOS rules are very similar to the CCS ones, apart from the labels that record the link chains involved in the transitions: moving from dyadic to *linked* interaction does not introduce any complexity burden. We comment in details the rules *(Act)*, *(Res)*, and *(Com)*. In rules *(Res)* and *(Com)* we leave implicit the side conditions $(\nu\, a)s \neq \bot$ and $s \bullet s' \neq \bot$, respectively (they can be easily recovered by noting that otherwise the label of the transition in the conclusion would be undefined).

The rule *(Act)* states that $\ell.P \xrightarrow{s} P$ for any simple link chain s whose unique solid link is ℓ. Intuitively, $\ell.P$ can take part in any interaction, in any (admissible) position. To join in a communication, $\ell.P$ should suitably enlarge its link ℓ to a simple link chain s including it, whose length is the same for all participants in order to proceed with the merge operation. Following the early style, the suitable length can be inferred at the time of deducing the input transition. Note that if one end of ℓ is τ, then ℓ can only appear at one extreme of s.

In the *(Res)* rule, the operator (νa) applied to s, can serve different aims: *floating*, if a does not appear in s, then $(\nu\, a)s = s$; *hiding*, if a is matched in s (i.e., a appears as ends already matched by adjacent links), then all occurrences of a in s are renamed to τ in $(\nu\, a)s$; *blocking*, if a is pending in s (i.e., there is a 'non-matched' occurrence of a in s), then $(\nu\, a)s = \bot$ and the rule cannot be applied.

In the *(Com)* rule the link chains recorded on both the premises' transitions are merged in the conclusion's transition. This is possible only if s and s' are to some extent "complementary". Contrary to CCS, the rule *(Com)* can be applied several times to prove that a transition is possible, because $s \bullet s'$ can still contain virtual links (if s and s' had a virtual link in the same position). However, when $s \bullet s'$ is solid, no further synchronisation is possible (by Lemma 1 (ii)).

$$\frac{\ell \in s \quad (\ell \textit{ only} \text{ solid link in } s)}{\ell.P \xrightarrow{s} P}\ (\text{Act}) \qquad \frac{P \xrightarrow{s} P'}{P+Q \xrightarrow{s} P'}\ (\text{Lsum}) \qquad \frac{P \xrightarrow{s} P'}{(\nu\, a)P \xrightarrow{(\nu\, a)s} (\nu\, a)P'}\ (\text{Res})$$

$$\frac{P \xrightarrow{s} P'}{P|Q \xrightarrow{s} P'|Q}\ (\text{Lpar}) \qquad \frac{P \xrightarrow{s} P' \qquad Q \xrightarrow{s'} Q'}{P|Q \xrightarrow{s \bullet s'} P'|Q'}\ (\text{Com}) \qquad \frac{P[X \mapsto \mathsf{rec}X.\,P] \xrightarrow{s} P'}{\mathsf{rec}X.\,P \xrightarrow{s} P'}\ (\text{Rec})$$

Fig. 1. SOS semantics of the CNA (rules (*Rsum*) and (*Rpar*) omitted)

Example 3. Let $P = {}^{\tau}\backslash_{a}.P_1|(\nu\, b)Q$ and $Q = {}^{b}\backslash_{\tau}.P_2|{}^{a}\backslash_{b}$. The process ${}^{\tau}\backslash_{a}.P_1$ can output on a, while ${}^{b}\backslash_{\tau}.P_2$ can input from b; the process ${}^{a}\backslash_{b}$ provides a one-shot link forwarder from a to b. Together, they can synchronise by agreeing to form a solid link chain of length 3, as follows, where $(\nu b)^*\backslash^{a}_{*}\backslash^{b}_{b}\backslash_{\tau} = {}^*\backslash^{a}_{*}\backslash^{\tau}_{\tau}\backslash_{\tau}$.

$$\dfrac{\dfrac{}{P \xrightarrow{{}^{\tau}\backslash^{a}_{a}\backslash^{\tau}_{\tau}\backslash_{\tau}} P_1|(\nu\, b)(P_2|\mathbf{0})}\ (\text{Com})}{\dfrac{}{{}^{\tau}\backslash_{a}.P_1 \xrightarrow{{}^{\tau}\backslash^{*}_{a}\backslash^{*}_{*}\backslash_{*}} P_1}\ (\text{Act}) \qquad \dfrac{Q \xrightarrow{{}^{*}\backslash^{a}_{*}\backslash^{b}_{b}\backslash_{\tau}} P_2|\mathbf{0}}{(\nu\, b)Q \xrightarrow{{}^{*}\backslash^{a}_{*}\backslash^{\tau}_{\tau}\backslash_{\tau}} (\nu\, b)(P_2|\mathbf{0})}\ (\text{Res})}\ (\text{Com})$$

$$\dfrac{\dfrac{}{{}^{b}\backslash_{\tau}.P_2 \xrightarrow{{}^{*}\backslash^{*}_{*}\backslash^{b}_{*}\backslash_{\tau}} P_2}\ (\text{Act}) \qquad \dfrac{}{{}^{a}\backslash_{b}.\mathbf{0} \xrightarrow{{}^{*}\backslash^{a}_{*}\backslash^{*}_{b}\backslash_{*}} \mathbf{0}}\ (\text{Act})}{Q \xrightarrow{{}^{*}\backslash^{a}_{*}\backslash^{b}_{b}\backslash_{\tau}} P_2|\mathbf{0}}\ (\text{Com})$$

The following lemma, whose proof goes by rule induction, shows that labels behave like an accordion. Any label s in a transition is interchangeable or replaceable with any chain having one or more ${}^*\backslash_*$ either on the left or on the right or on both sides of s. It is also replaceable with any composition where each ${}^*\backslash_*$ inside s is replaced by one or more ${}^*\backslash_*$. This fact can be exploited later, when the abstract semantics is given.

Lemma 3

(i) If $P \xrightarrow{s} P'$ and $s^*\backslash_*$ (resp. ${}^*\backslash_* s$) is valid, then $P \xrightarrow{s^*\backslash_*} P'$ (resp. $P \xrightarrow{{}^*\backslash_* s} P'$). Vice versa, if $P \xrightarrow{s'} P'$ and $s' = s^*\backslash_*$ or $s' = {}^*\backslash_* s$, then $P \xrightarrow{s} P'$.

(ii) If $P \xrightarrow{{s_1}^*\backslash_* s_2} P'$ then $P \xrightarrow{{s_1}^*\backslash^{*}_{*}\backslash_* s_2} P'$. Vice versa, if $P \xrightarrow{{s_1}^*\backslash^{*}_{*}\backslash_* s_2} P'$ then $P \xrightarrow{{s_1}^*\backslash_* s_2} P'$.

(iii) If $P \xrightarrow{{s_1}^{\alpha}\backslash^{a}_{a}\backslash_{\beta} s_2} P'$ then $P \xrightarrow{{s_1}^{\alpha}\backslash^{*}_{a}\backslash^{a}_{*}\backslash_{\beta} s_2} P'$. Vice versa, if $P \xrightarrow{{s_1}^{\alpha}\backslash^{*}_{a}\backslash^{a}_{*}\backslash_{\beta} s_2} P'$ then $P \xrightarrow{s={s_1}^{\alpha}\backslash^{a}_{a}\backslash_{\beta} s_2} P'$.

Note that if $P \xrightarrow{{}^{a}\backslash^{\tau}_{\tau}\backslash_{b}} P'$, then it is not the case that a virtual link can be inserted in the middle of the chain, because the τ's may represent a communication on a restricted channel and ${}^{a}\backslash^{*}_{\tau}\backslash^{\tau}_{*}\backslash_{b}$ is not a valid link chain anyway.

Routing examples. We give a few examples to show how flexible is CNA for defining "routing" policies. We have already seen a one-shot, one-hop forwarder from a to b, that can be written as ${}^{a}\backslash_{b}.\mathbf{0}$. Its persistent version is just written as $P^{a}_{b} \triangleq \mathsf{rec}X.\,{}^{a}\backslash_{b}.X$. Moreover, with $P^{a}_{b}\ |P^{b}_{a}$ we obtain a sort of name fusion

between a and b, i.e., a and b can be interchangeably used. An alternating forwarder from a to b first and then to c can be defined as $A^a_{b,c} \triangleq \mathsf{rec}X.\,({}^{a}\backslash_{b}.{}^{a}\backslash_{c}.X)$. A persistent non-deterministic forwarder, from a to $c_1...c_n$ can be written, e.g., as $P^a_{c_1...c_n} \triangleq \mathsf{rec}X.\,({}^{a}\backslash_{c_1}.X + \cdots + {}^{a}\backslash_{c_n}.X)$. Similarly, $P^{b_1...b_m}_a \triangleq \mathsf{rec}X.\,({}^{b_1}\backslash_{a}.X + \cdots + {}^{b_m}\backslash_{a}.X)$ is a persistent non-deterministic forwarder, from $b_1...b_m$ to a. By combining the two processes as $(\nu\, a)(P^{b_1...b_m}_a | P^a_{c_1...c_n})$, then we obtain a persistent forwarder from any of the b_i's to any of the c_j's.

Abstract semantics. As usual, we can use the LTS semantics to define suitable behavioural equivalences over processes. We are interested in bisimilarity. However, when comparing two labels we abstract away from the number and identities of hops performed and from the size of the sequences of virtual links.

Definition 1. *We let $\bowtie$ be the least equivalence relation over link chains closed under the following axioms (remind that $\alpha, \beta, \gamma \in \mathcal{C} \cup \{\tau, *\}$):*

$$ {}^{*}\backslash_{*} s \bowtie s \qquad s_1 {}^{\alpha}\backslash^{\gamma}_{\gamma}\backslash_{\beta} s_2 \bowtie s_1 {}^{\alpha}\backslash_{\beta} s_2 \qquad s^{*}\backslash_{*} \bowtie s $$

A link chain is *essential* if it is composed by alternating solid and virtual links, and has solid links at its ends. An essential link chain has minimal length with respect to equivalent link chains and concisely represents the missing "paths" of interactions between those already completed. For example, we have that ${}^{*}\backslash^{a}_{*}\backslash^{\tau}_{\tau}\backslash^{*}_{b}\backslash^{*}_{*}\backslash^{c}_{*}\backslash^{d}_{d}\backslash^{*}_{a}\backslash_{*}$ is equivalent to the essential chain ${}^{a}\backslash^{*}_{b}\backslash^{c}_{*}\backslash_{a}$.

Lemma 4
(i) Let s and s' be two essential link chains s.t. $s \bowtie s'$, then $s = s'$.
(ii) Let s be an essential link chain. For any $s' \bowtie s$ we have $|s'| \geq |s|$.

It is immediate to check that by orienting the axioms in Def. 1 from left to right we have a procedure to transform any link chain s to a unique essential link chain s' such that $s \bowtie s'$. We write $\mathsf{e}(s)$ to denote such unique representative, which enjoys the following nice properties, useful in the proof of the following Proposition 1. The first part of the lemma says that $\mathsf{e}(\cdot)$ induces an equivalence over link chains that is a congruence with respect to juxtaposition of link chains (when it is well-defined). The second part says that taken two link chains s_1 and s_2 in the same equivalence class, and given any sequence s that can be merged with s_1, then it is possible to find a link chain s', in the same equivalence class as s, such that it can be merged with s_2 and the result is equivalent to $s \bullet s_1$.

Lemma 5
(i) If $\mathsf{e}(s_1) = \mathsf{e}(s_2)$, then for any s such that $s_1 s$ (resp. ss_1) is a link chain we have that also $s_2 s$ (resp. ss_2) is a link chain and $\mathsf{e}(s_1 s) = \mathsf{e}(s_2 s)$ (resp. $\mathsf{e}(ss_1) = \mathsf{e}(ss_2)$).
(ii) If $\mathsf{e}(s_1) = \mathsf{e}(s_2)$, then for any s such that $s \bullet s_1 \neq \bot$ there exists a link chain $s' \bowtie s$ such that $s' \bullet s_2 \neq \bot$ and $\mathsf{e}(s \bullet s_1) = \mathsf{e}(s' \bullet s_2)$.

In the following, we write $P \to P'$ when $P \xrightarrow{s} P'$ for some s s.t. $\mathsf{e}(s) = {}^{\tau}\backslash_{\tau}$.

Definition 2. *A* network bisimulation $\mathbf{R}$ *is a binary relation over CNA processes such that, if* $P\ \mathbf{R}\ Q$ *then:*

- *if* $P \xrightarrow{s} P'$, *then* $\exists\ s', Q'$ *such that* $\mathsf{e}(s) = \mathsf{e}(s')$, $Q \xrightarrow{s'} Q'$, *and* $P'\ \mathbf{R}\ Q'$;
- *if* $Q \xrightarrow{s} Q'$, *then* $\exists\ s', P'$ *such that* $\mathsf{e}(s) = \mathsf{e}(s')$, $P \xrightarrow{s'} P'$, *and* $P'\ \mathbf{R}\ Q'$.

We let $\sim_n$ denote the largest network bisimulation and we say that P is *network bisimilar* to Q if $P \sim_n Q$.

Example 4. Consider the two processes $P \triangleq \mathsf{rec}X.\,{}^{a}\backslash_{b}.X$ and $Q \triangleq \mathsf{rec}X.\,(\nu\, c)\,({}^{a}\backslash_{c} \mid {}^{c}\backslash_{b}.X)$. We have that whenever $P \xrightarrow{s} P'$, then $P' = P$ and $\mathsf{e}(s) = {}^{a}\backslash_{b}$. Similarly, whenever $Q \xrightarrow{s} Q'$, then $Q' = (\nu\, c)(\mathbf{0}|Q)$ and $\mathsf{e}(s) = {}^{a}\backslash_{b}$. Then we prove that $P \sim_n Q$ by showing that the relation $\mathbf{R}$ below:

$$\mathbf{R} \triangleq \{(P, R) \mid \exists n. R = C^n[Q]\}$$

is a network bisimulation, where $C^n[Q]$ is inductively defined by letting $C^0[Q] \triangleq Q$ and $C^{n+1}[Q] \triangleq C[C^n[Q]]$ for $C[\cdot]$ the context $(\nu\, c)(\mathbf{0}|\cdot)$

Proposition 1. *Network bisimilarity is a congruence.*

Proof. The proof uses standard arguments. The only non-trivial case is that of parallel composition. We want to prove that if $P \sim_n Q$ then for any R we have $P|R \sim_n Q|R$. We define the relation $\mathbf{R}_| \triangleq \{(P|R, Q|R) \mid P \sim_n Q\}$ and show that $\mathbf{R}_|$ is a bisimulation. Suppose $P \sim_n Q$ and $P|R \xrightarrow{s} T$. We want to prove that $Q|R \xrightarrow{s} T'$ with $T\ \mathbf{R}_|\ T'$. There are three cases to be considered, depending on the last SOS rule used to prove $P|R \xrightarrow{s} T$. If the used rule is

- *(Rpar)*, then it means that $R \xrightarrow{s} R'$ for some R' with $T = P|R'$. But then, by using *(Rpar)* we have $Q|R \xrightarrow{s} Q|R'$ and $P|R'\ \mathbf{R}_|\ Q|R'$ by definition of $\mathbf{R}_|$.
- *(Lpar)*, then it means that $P \xrightarrow{s} P'$ for some P' with $T = P'|R$. By assumption we know that $P \sim_n Q$ and therefore there exists s', Q' s.t. $Q \xrightarrow{s'} Q'$ with $\mathsf{e}(s) = \mathsf{e}(s')$ and $P' \sim_n Q'$. By applying the rule *(Lpar)* we have that $Q|R \xrightarrow{s'} Q'|R$ and we have done because $P'|R\ \mathbf{R}_|\ Q'|R$ by definition of $\mathbf{R}_|$.
- *(Com)*, then it means that $P \xrightarrow{s_1} P'$, $R \xrightarrow{s_2} R'$, for some s_1, s_2, P', R' with $s = s_1 \bullet s_2$ and $T = P'|R'$. By assumption we know that $P \sim_n Q$ and therefore there exists s_1', Q' s.t. $Q \xrightarrow{s_1'} Q'$ with $\mathsf{e}(s_1) = \mathsf{e}(s_1')$ and $P' \sim_n Q'$. Now it may be the case that $s_1' \bullet s_2$ is not defined, but by the previous technical lemmata, we know that s_1' and s_2 can be stretched resp. to s_1'' and s_2' by inserting enough virtual links to have that $s_1'' \bullet s_2'$ is defined, $\mathsf{e}(s_1 \bullet s_2) = \mathsf{e}(s_1'' \bullet s_2')$, $Q \xrightarrow{s_1''} Q'$ and $R \xrightarrow{s_2'} R'$. We conclude by applying rule *(Com)*: $Q|R \xrightarrow{s'} Q'|R'$ with $P'|R'\ \mathbf{R}_|\ Q'|R'$ by definition of $\mathbf{R}_|$.

Analogously to CCS, it is immediate to check that several useful axioms over processes hold up to network bisimilarity, like the commutative monoidal laws for $|$ and $+$, the idempotence of $+$ and the usual laws about restriction.

2 The Calculus of Linked Interactions

We can now enhance the algebra to deal with name passing and call it the *calculus of linked interactions* (`link`-*calculus* for short), by extending, in the syntax, link prefixes with tuples of arguments, where each link in the whole chain simply carries the same list of arguments, but with different (send/receive) capabilities.

$$P, Q ::= \cdots \mid \ell t.P$$

In this way, we keep apart the interaction mechanism from the name-passing one that eventually fit together in synchronisations. We have just borrowed from π-calculus the name handling machinery (and liberated it from dyadic interaction legacy), still having input, output and extrusion mechanisms.

In the tuple $t = \langle \vec{w} \rangle$, names can be used either as values or as variables. To be distinguished, variables are underlined. During communication, variables are instantiated by values, while values are used for matching arguments, as in [1].

Example 5. Consider e.g., the two tuples in two "complementary" prefixes like ${}^{\tau}\backslash_{a}\langle id, n, \underline{x}\rangle.P$ and ${}^{a}\backslash_{\tau}\langle id, \underline{y}, m\rangle.Q$, where $\underline{x}$ is an input for P, $\underline{y}$ is an input for Q, and id is a name known by both processes: the two links can be merged, the first parameters must match exactly, n is assigned to y, while m is assigned to x.

This mechanism allows, e.g., a form of multi-way communication, where all peers involved in the chain link can express arguments to be matched and provide actual arguments to replace the formal ones of other peers. For a tuple t, we let $vals(t)$ and $vars(t)$ denote the set of values and the set of variables of t, respectively. We say that a tuple t is *ground* if $vars(t) = \emptyset$.

We assume action names are admissible values, i.e., as in π-calculus we have the possibility to communicate (names of) means of communication. Similarly to $(\nu\, a)$, the prefix $\ell t.P$ binds the occurrences of the variables $vars(t)$ in P (and the notions of free names $fn(P)$, bound names $bn(P)$ and alpha-conversion are updated accordingly). In the following, given two sets of names S and T, we write $S\#T$ as a shorthand for $S \cap T = \emptyset$.

Operational semantics. The operational semantics is defined in terms of an LTS whose states are `link`-calculus processes, whose labels are pairs sg of link chains and tuples and whose transitions are generated by the SOS rules in Fig. 2.

In a label of the form sg, with s solid, g must be the empty tuple $\langle\rangle$, i.e., it is not possible to observe the arguments of a completed communication. We let s abbreviate $s\langle\rangle$.

We denote by $\sigma = [x_1 \mapsto v_1, ..., x_n \mapsto v_n]$ the substitution that replaces each x_i with v_i, and is the identity otherwise, and set $vars(\sigma) = \{ x_1, ..., x_n \}$. For a tuple $t = \langle w_1, ..., w_n \rangle$, a link $\ell = {}^{\alpha}\backslash_{\beta}$, and a substitution $\sigma = [x_1 \mapsto v_1, ..., x_n \mapsto v_n]$ we define $t\sigma$ and $\ell\sigma$ element-wise as:

$$t\sigma \triangleq \langle w_1\sigma, ..., w_n\sigma \rangle \qquad a\sigma \triangleq \begin{cases} v_i & \text{if } a = x_i \text{ for some } i \in [1,n] \\ a & \text{otherwise} \end{cases}$$

$$\ell\sigma \triangleq {}^{\alpha\sigma}\backslash_{\beta\sigma} \qquad \underline{a}\sigma \triangleq \begin{cases} \underline{v_i} & \text{if } a = x_i \text{ for some } i \in [1,n] \\ \underline{a} & \text{otherwise} \end{cases}$$

The application of σ on processes is defined as below (note that, as processes are taken up to alpha-conversion of bound names, it is always possible to find suitable representatives of $\ell t.P$ and $(\nu\, a)P$, such that $(\ell t.P)\sigma$ and $((\nu\, a)P)\sigma$ are well-defined):

$$\begin{array}{rl} \mathbf{0}\sigma \triangleq \mathbf{0} & (\ell t.P)\sigma \triangleq (\ell\sigma)(t\sigma).(P\sigma), \text{if } vars(t)\#(vars(\sigma) \cup vals(\sigma)) \\ (P+Q)\sigma \triangleq P\sigma + Q\sigma & \\ (P|Q)\sigma \triangleq P\sigma|Q\sigma & ((\nu\, a)P)\sigma \triangleq (\nu a)P\sigma, \text{if } \{a\}\#(vars(\sigma) \cup vals(\sigma)) \\ X\sigma \triangleq X & (\mathsf{rec}X.\, P)\sigma \triangleq \mathsf{rec}X.\, (P\sigma) \end{array}$$

We say that g is a *full instance* of t and write $g \preceq_\sigma t$ if $vars(\sigma) = vars(t) \wedge g = t\sigma$.

Like in the π-calculus, names in the tuple can be extruded during the communication. In the labels of transitions, we need to annotate positions in the tuple to distinguish between arguments that are taken in input (i.e., they are guessed instances), or that are extruded. We underline the former and overline the latter. A name can be extruded when it is not already annotated; after the extrusion, it will be overlined. Formally, given a (annotated) tuple g, we define $(\nu\, a)sg$ and $(\nu\, a)g$ as follows:

$$(\nu\, a)(sg) \triangleq ((\nu\, a)s)((\nu\, a)g) \qquad (\nu\, a)\langle w_1, ..., w_n\rangle \triangleq \langle (\nu\, a)w_1, ..., (\nu\, a)w_n\rangle \qquad (\nu\, a)w \triangleq \begin{cases} w & \text{if } w \neq a, \overline{a}, \underline{a} \\ \overline{a} & \text{if } w = a \end{cases}$$

We let $ex(g)$ denote the set of extruded (i.e., overlined) names appearing in g. We write $a \in g$ if the name a appears in the tuple g (with or without annotation).

Lemma 6. *If $(\nu\, a)g \neq \bot$, then $vars((\nu\, a)g) = vars(g)$.*

Two annotated tuples can be merged when they list exactly the same values in the same order, and if the values in matching positions are annotated in some compatible way. Formally, if $\langle \vec{w}\rangle = \langle w_1, ..., w_n\rangle$ and $\langle \vec{w'}\rangle = \langle w'_1, ..., w'_n\rangle$:

$$sg \bullet s'g' \triangleq (s \bullet s')(g \bullet g') \qquad \langle \vec{w}\rangle \bullet \langle \vec{w'}\rangle \triangleq \langle w_1 \bullet w'_1, ..., w_n \bullet w'_n\rangle$$

$$w \bullet w' \triangleq \begin{cases} w & \text{if } (w = w' = v) \vee (w = w' = \underline{v}) \\ v & \text{if } (w = v \wedge w' = \underline{v}) \vee (w = \underline{v} \wedge w' = v) \\ \overline{v} & \text{if } (w = \overline{v} \wedge w' = \underline{v}) \vee (w = \underline{v} \wedge w' = \overline{v}) \end{cases}$$

Example 6. Back to Ex. 5, ${}^{\tau}\backslash^{*}_{a}\backslash_{*}\langle id, n, \underline{m}\rangle \bullet {}^{*}\backslash^{a}_{*}\backslash_{\tau}\langle id, \underline{n}, m\rangle = ({}^{\tau}\backslash^{*}_{a}\backslash_{*} \bullet {}^{*}\backslash^{a}_{*}\backslash_{\tau})(\langle id, n, \underline{m}\rangle \bullet \langle id, \underline{n}, m\rangle) = {}^{\tau}\backslash^{a}_{a}\backslash_{\tau}\langle id \bullet id, n \bullet \underline{n}, \underline{m} \bullet m\rangle = {}^{\tau}\backslash^{a}_{a}\backslash_{\tau}\langle id, n, m\rangle$. Recall that in the early-style (Act) rule the values to be received are guessed and so the prefix variables $\underline{y}$ and $\underline{x}$ are replaced by $\underline{n}$ and $\underline{m}$, respectively.

Lemma 7. *If $g \bullet g' \neq \bot$, then $vars(g \bullet g') \subseteq vars(g) \cap vars(g')$.*

A close look at the SOS rules in Fig. 2 shows that they resemble the early semantic rules of π-calculus. The main difference is that we are dealing with a multi-party form of communication, hence the "close" rule must be applied when the communication has been completed. Let us briefly comment on the rules.

Rule *(Act)* allows the process $\ell t.P$ to offer the tuple t in a communication on the link ℓ. More precisely, following an early style, the actual tuple to be communicated

$$\frac{\ell \in s \quad (\ell \textit{ only} \text{ solid link in } s) \qquad g \preceq_\sigma t}{\ell t.P \xrightarrow{sg} P\sigma}\ (\text{Act}) \qquad \frac{P \xrightarrow{sg} P'}{P + Q \xrightarrow{sg} P'}\ (\text{Lsum})$$

$$\frac{P[X \mapsto \mathsf{rec}X.\,P] \xrightarrow{sg} P'}{\mathsf{rec}X.\,P \xrightarrow{sg} P'}\ (\text{Rec}) \qquad \frac{P \xrightarrow{sg} P' \qquad ex(g)\#fn(Q)}{P|Q \xrightarrow{sg} P'|Q}\ (\text{Lpar})$$

$$\frac{P \xrightarrow{sg} P' \qquad a \notin g}{(\nu\, a)P \xrightarrow{(\nu\, a)sg} (\nu\, a)P'}\ (\text{Res}) \qquad \frac{P \xrightarrow{sg} P' \qquad a \in g}{(\nu\, a)P \xrightarrow{(\nu\, a)sg} P'}\ (\text{Open})$$

$$\frac{P \xrightarrow{sg} P' \qquad Q \xrightarrow{s'g'} Q' \qquad \begin{array}{l} ex(g)\#fn(Q) \\ ex(g')\#fn(P) \end{array} \qquad s \bullet s' \text{ is not solid}}{P|Q \xrightarrow{sg \bullet s'g'} P'|Q'}\ (\text{Com})$$

$$\frac{P \xrightarrow{sg} P' \qquad Q \xrightarrow{s'g'} Q' \qquad \begin{array}{ll} ex(g)\#fn(Q) & s \bullet s' \text{ is solid} \\ ex(g')\#fn(P) & g \bullet g' \text{ is ground} \end{array}}{P|Q \xrightarrow{s \bullet s'} (\nu\, ex(g \bullet g'))(P'|Q')}\ (\text{Close})$$

Fig. 2. SOS semantics of the `link`-calculus (rules (*Rsum*) and (*Rpar*) omitted)

must be a full instance of t (see the condition $g \preceq_\sigma t$): the communication is that of sg, where variables appearing in t are replaced in $g = t\sigma$ by actual parameters. The substitution σ is also applied to the continuation, after the transition ($P\sigma$).

In rules *(Res)* and *(Open)* we leave implicit the side condition $(\nu\, a)sg \neq \bot$. Rule *(Res)* is applicable whenever $a \notin g$, in which case $(\nu\, a)g = g \neq \bot$ and thus $(\nu\, a)(sg) = ((\nu\, a)s)g$. Rule *(Open)* models the extrusion of a. Note that, since $(\nu\, a)sg \neq \bot$ and $a \in g$, we have that the only possibility for a to appear in g is without annotations (otherwise $(\nu\, a)g = \bot$). Then, by definition of $(\nu\, a)g$, all occurrences of a within g are overlined in $(\nu\, a)g$ (to denote the name extrusion).

In rule *(Com)*, the annotated tuples are "complementary" and can be merged, by merging the link chains and the two tuples. Note that we leave implicit the side condition $sg \bullet s'g' \neq \bot$, because the $sg \bullet s'g'$ annotates the label of the conclusion transition. In addition, rule *(Com)* checks that the extruded names of one process do not clash with the free names of the other process (like in ordinary π-calculus) and finally that the communication is not completed yet ($s \bullet s'$ not solid).

Rule *(Close)* premises differ from *(Com)* one, because *(Close)* is applicable only when the communication has been fully completed and cannot be further extended ($s \bullet s'$ is solid), in which case we must close, i.e., put back the restriction of all names extruded in the communication ($\nu\, ex(g \bullet g'')$). Still, we need to make sure that $g \bullet g'$ has no unresolved input, i.e., that all requested values have been issued ($g \bullet g'$ is ground). Moreover, the observed label is just $s \bullet s'$, as explained before. This is similar to the π-calculus mechanism, according to which the synchronisation of e.g., $a\langle x\rangle$ and $\overline{a}\langle x\rangle$ yields τ and not $\tau\langle x\rangle$.

While rules *(Lsum)*, *(Rsum)* and *(Rec)* are straightforward, rules *(Lpar)* and *(Rpar)* need just to check that extruded names of one process do not clash with free names of the other process (like in ordinary π-calculus).

Note that if the only used tuples are the empty ones $\langle\rangle$, then the semantics rules coincide with the ones of CNA.

Example 7. Consider the following restricted process, built on the processes in Ex. 5, $S \triangleq (\nu\, a)({}^{\tau}\backslash_{a}\langle id, n, \underline{x}\rangle.P | (\nu\, m)^{a}\backslash_{\tau}\langle id, \underline{y}, m\rangle.Q)$. In one step, S can reduce to $S' \triangleq (\nu\, a)(\nu\, m)(P[x \mapsto m] | Q[y \mapsto n])$, via the communication ${}^{\tau}\backslash^{\tau}_{\tau}\backslash_{\tau}\langle id, n, \overline{m}\rangle$, where the restriction on a hides the matched occurrences of a in ${}^{\tau}\backslash^{a}_{a}\backslash_{\tau}$ and the restriction on m causes the extrusion of the name m (to process P). Moreover, since the chain link ${}^{\tau}\backslash^{\tau}_{\tau}\backslash_{\tau}$ is solid (i.e., it cannot be extended further) and the tuple $\langle id, n, \overline{m}\rangle$ is ground, we remove the tuple from the observed label, i.e., $S \xrightarrow{{}^{\tau}\backslash^{\tau}_{\tau}\backslash_{\tau}} S'$. No other interaction is possible.

Abstract semantics. By analogy with the early bisimilarity of the π-calculus, we extend the notion of network bisimilarity to consider the tuples of names.

Definition 3. *A* linked bisimulation $\mathbf{R}$ *is a binary relation over* `link`*-calculus processes such that, if* $P\ \mathbf{R}\ Q$ *then:*

- *if* $P \xrightarrow{sg} P'$ *with* $ex(g)\#fn(P)$*, then there exists* s' *and* Q' *such that* $\mathtt{e}(s) = \mathtt{e}(s')$*,* $Q \xrightarrow{s'g} Q'$*, and* $P'\ \mathbf{R}\ Q'$*;*
- *if* $Q \xrightarrow{sg} Q'$ *with* $ex(g)\#fn(Q)$*, then there exists* s' *and* P' *such that* $\mathtt{e}(s) = \mathtt{e}(s')$*,* $P \xrightarrow{s'g} P'$*, and* $P'\ \mathbf{R}\ Q'$*.*

We let $\sim_l$ denote the largest linked bisimulation and we say that P is *linked bisimilar* to Q if $P \sim_l Q$.

The following result may look surprising, since early bisimilarity is not a congruence in the case of π-calculus, due to the input prefix context. The classic counterexample is $P = x|\overline{y}$ being bisimilar to $Q = x.\overline{y} + \overline{y}.x$ but $a(y).P$ being not bisimilar to $a(y).Q$ (when the name received on y is x, a τ move is available for $P[y \mapsto x]$ but not for $Q[y \mapsto x]$). The fact is that the SOS semantics of the `link`-calculus can collect the ready set of prefixes concurrently available to be executed (e.g., by separating them through virtual links); thus the above P and Q are not considered as bisimilar. In other words, virtual links allow to establish an interaction between different ends, as if a substitution was available to rename one end into the other, i.e., the semantics is already substitution closed.

Proposition 2. *Linked bisimilarity is a congruence.*

3 Background on Mobile Ambients

In this section, we briefly recall Mobile Ambients (MA) syntax and semantics, in order to show the encoding of MA in the `link`-calculus, in the next section.

$$\frac{}{n[\,\mathsf{in}\,m.P \mid Q\,] \mid m[\,R\,] \to m[\,n[\,P \mid Q\,] \mid R\,]}\ (\text{In})$$

$$\frac{}{m[\,n[\,\mathsf{out}\,m.P \mid Q\,] \mid R\,] \to n[\,P \mid Q\,] \mid m[\,R\,]}\ (\text{Out})$$

$$\frac{}{\mathsf{open}\,n.P \mid n[\,Q\,] \to P \mid Q}\ (\text{Open}) \qquad \frac{P \to Q}{(\nu\, n)P \to (\nu\, n)Q}\ (\text{Res}) \qquad \frac{P \to Q}{n[\,P\,] \to n[\,Q\,]}\ (\text{Amb})$$

$$\frac{P \to Q}{P \mid R \to Q \mid R}\ (\text{Par}) \qquad \frac{P' \equiv P \qquad P \to Q \qquad Q \equiv Q'}{P' \to Q'}\ (\text{Cong})$$

$P \equiv P$	$Q \equiv P \Rightarrow P \equiv Q$	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
$P \mid \mathbf{0} \equiv P$	$P \mid Q \equiv Q \mid P$	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$(\nu\, n)\mathbf{0} \equiv \mathbf{0}$	$(\nu\, n)(\nu\, m)P \equiv (\nu\, m)(\nu\, n)P$	$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$
	$(\nu\, n)(P \mid Q) \equiv P \mid (\nu\, n)Q$, if $n \notin fn(P)$	$P \equiv Q \Rightarrow (\nu\, n)P \equiv (\nu\, n)Q$
$!P \equiv P \mid !P$	$(\nu\, n)(m[\,P\,]) \equiv m[\,(\nu\, n)P\,]$, if $n \neq m$	$P \equiv Q \Rightarrow n[\,P\,] \equiv n[\,Q\,]$

Fig. 3. Reduction and Structural Congruence Rules for the Mobile Ambients

Mobile Ambients (MA) [9] is a calculus for *mobility* that includes both mobile agents and mobile computational ambients in which agents interact. Ambients are *bounded* locations, such as a web page or a virtual address space, that can be moved as a whole. Each ambient has a name and can include sub-ambients to form a hierarchical structure, that can be dynamically modified by agents.

We focus here on the so-called *pure* MA, i.e., disregarding communication primitives and variables for brevity. However our results in Section 4 can be easily extended to the more general case.

Let n range over the numerable set of names $\mathcal{N}$. The set of mobile ambient processes $\mathcal{P}_{MA}$ (with metavariable P) and the set of capabilities $\mathcal{C}ap$ (with metavariable M) are defined below:

$$P ::= \mathbf{0} \mid (\nu\, n)P \mid P|Q \mid !P \mid n[\,P\,] \mid M.P$$
$$M ::= \mathsf{in}\, n \mid \mathsf{out}\, n \mid \mathsf{open}\, n$$

The first four constructs are quite standard in process calculi, while the other ones are specific to ambients: $n[\,P\,]$ denotes the ambient n in which the process P runs; $M.P$ executes an action depending on the capability M and then behaves as P. There are three kinds of capabilities: one for entering, one for exiting and one for opening up an ambient. In the process $\mathsf{in}\, m.P$ the entry capability allows the immediately surrounding ambient n (if any) to find a sibling ambient named m where to enter (i.e., to become a child of m). Similarly, in the process $\mathsf{out}\, m.P$ the exit capability allows the immediately surrounding ambient n to exit its parent ambient (if named m) and to become a sibling of m. Finally, in $\mathsf{open}\, m.P$ the open capability allows the boundary of an ambient m located in parallel to be dissolved. Each interaction involves more than two parties.

The only binder is $(\nu\, n)$ and the sets of free names and of bound names of a process P are defined in the obvious way and denoted, respectively, by $fn(P)$ and $bn(P)$. As usual, the restriction of a sequence of names $\vec{m} = \{m_1, ..., m_k\}$ for a process P is denoted as $(\nu\vec{m})P$ and stands for $(\nu m_1)...(\nu m_k)P$. We shall denote by $\sigma = [n \mapsto m]$ the (capture-avoiding) substitution that replaces n by m, and by $P\sigma$ the process obtained by applying σ to P to replace all free occurrences of n in P by m. Processes are taken up to alpha-conversion of restricted names, i.e., $(\nu\, n)P$ denotes the same process as $(\nu\, m)(P[n \mapsto m])$ whenever $m \notin fn(P)$.

The semantics of the MA is given by the reduction and the structural congruence rules in Fig. 3. Besides the one-step reduction rules for movement capabilities ((In), (Out) and $(Open)$) and the usual reduction rule for congruence $(Cong)$, the other rules propagate reductions across scopes (Res), ambient nesting (Amb) and parallel composition (Par).

Example 8. Consider the process $P \triangleq m[\, s[\, \mathsf{in}\, n.R\,] \mid T\,] \mid \mathsf{open}\, m.Q \mid n[\, G\,]$ (for suitable R, T, Q, G) that can execute an $\mathsf{open}\, m$ followed by an $\mathsf{in}\, n$:

$$P \to s[\, \mathsf{in}\, n.R\,] \mid T \mid Q \mid n[\, G\,] \to T \mid Q \mid n[\, s[\, R\,] \mid G\,]$$

We write $P \downarrow_n$, and say that P *has barb* n, if $P \equiv (\nu\, \vec{m})(n[P_1]|P_2)$ for some names $\vec{m}$ and processes P_1 and P_2 with $n \notin \vec{m}$. We say that P *has weak barb* n, written $P \Downarrow_n$ if there exists P' such that $P \to^* P'$ and $P' \downarrow_n$, for $\to^*$ the reflexive and transitive closure of the (immediate) reduction relation $\to$. Let $\mathbf{R}$ a binary relation on processes. We say that:

- $\mathbf{R}$ is *preserved by contexts* if $P\ \mathbf{R}\ Q$ implies $C[P]\ \mathbf{R}\ C[Q]$ for any context $C[\cdot]$;
- $\mathbf{R}$ is *closed under reductions* (or *reduction closed*) if whenever $P\ \mathbf{R}\ Q$ and $P \to P'$ then there exists Q' such that $Q \to^* Q'$ and $P'\ \mathbf{R}\ Q'$;
- $\mathbf{R}$ is *barb preserving* if $P\ \mathbf{R}\ Q$ and $P \downarrow_n$ implies $Q \Downarrow_n$.

Definition 4. Reduction barbed congruence, *written* $\cong$, *is the largest relation over processes, which is reduction closed, barb preserving, and preserved by all contexts.*

4 Encoding Mobile Ambients

We are now ready to show our encoding of MA in the `link`-calculus, that follows the idea developed in [5]. We pick MA as an interesting case because ambient interactions are inherently multi-party, even when they apparently involve only two parties. Each movement involves an ambient and the process exercising the corresponding capability, but the resulting reconfiguration of the hierarchical structure also impacts on the ambients and processes of the context.

Rule *(In)* requires a three-party interaction (at least), involving: 1) the process $\mathsf{in}\, m.P$ with the capability to enter the ambient m; 2) its parent ambient $n[\,\cdot\,]$ to be moved; 3) the ambient $m[\,\cdot\,]$ to be entered. The rule can be successfully applied only when all the three entities are available. Any encoding based on binary interactions must deal with atomicity issues in the completion of the

move, with conflicting, concurrent operations on the same ambient and with the possibility of retract/roll-back in case only two peers out of three are available.

Similarly, rule *(Out)* requires a three-party interaction (at least), involving: 1) the process $\mathsf{out}\, m.P$ with the capability to leave the ambient m; 2) its parent ambient $n[\,\cdot\,]$ to be moved; 3) the "grand-parent" ambient $m[\,\cdot\,]$ (where $n[\,\cdot\,]$ is enclosed) to be exited.

Rule *(Open)* apparently requires a two-party interaction only, but as a matter of fact it is more complex than the other two rules, as it introduces the need of multi-party interactions with an unbounded number of peers. This is because when the ambient $n[\,Q\,]$ is dissolved, its content Q must be relocated, which may consist of an unbounded (and not known a priori) number of parallel processes: they all participate to the interaction! We adopt here a syntactic solution with no semantic impact on the rest: we replace $n[\,\cdot\,]$ with some sort of blind forwarder that leaves Q unaware of the deletion of $n[\,\cdot\,]$. However, the presence of forwarders complicates the interactions needed by rules *(In)* and *(Out)*, because the three parties can now be connected via chains of forwarders of arbitrary length. Our forwarders are reminiscent of forwarders in π-calculus [15], inspired by [28].

Roughly, the idea is to define an encoding assigning to any MA process P a corresponding `link`-calculus process $[\![\, P \,]\!]_{\tilde{a}}$ (the role of names $\tilde{a}$ will be made clear later) such that:

- for any reduction $P \to P'$ there is a step $[\![\, P \,]\!]_{\tilde{a}} \to [\![\, P' \,]\!]_{\tilde{a}}$;
- and vice versa, for any silent step $[\![\, P \,]\!]_{\tilde{a}} \to Q$ there is an MA process P' such that $Q = [\![\, P' \,]\!]_{\tilde{a}}$ and $P \to P'$.

Unfortunately, a direct encoding has to deal with the presence of forwarders, so that in general:

- for any reduction $P \to P'$ we can find a step $[\![\, P \,]\!]_{\tilde{a}} \to Q$ but Q can differ from $[\![\, P' \,]\!]_{\tilde{a}}$ because of the presence of forwarders;
- and vice versa, for any silent step $[\![\, P \,]\!]_{\tilde{a}} \to Q$ we can find an MA process P' such that $P \to P'$ but, again, Q can differ from $[\![\, P' \,]\!]_{\tilde{a}}$ because of the presence of forwarders.

One possible turnaround is to show that the correspondence holds up to some suitable abstract equivalence instead of strict equality. Instead, we provide a tighter correspondence that introduces forwarders in the syntax of MA, with no effect whatsoever on the semantics and expressiveness, and that allows us to recover the stronger correspondence result sketched above, with exact matching between the reductions of MA and the silent steps of the `link`-calculus (modulo some standard structural laws imposed by the MA structural congruence).

Ambients within Brackets. We just extend the syntax of MA with the possibility to enclose a process P within a pair of parentheses:

$$P ::= \cdots \mid (\!|P|\!)$$

making the presence of parentheses inessential w.r.t the behaviour of the process. To this aim, we introduce the additional structural congruence axioms:

$$(\!|(\nu\, n)P|\!) \equiv (\nu\, n)(\!|P|\!) \qquad\qquad P \equiv Q \Rightarrow (\!|P|\!) \equiv (\!|Q|\!)$$

Finally, we define the notion of *passive context* $\mathbb{C}$, to adjust the basic reduction rules to deal with the presence of an arbitrary number of balanced parentheses.

$$\mathbb{C}, \mathbb{D}, \mathbb{E} ::= \bullet \mid (\!|\mathbb{C}|\!) \mid \mathbb{C}|P \mid P|\mathbb{C}$$

and write $\mathbb{C}(P)$ to denote the process obtained by replacing the hole $\bullet$ in $\mathbb{C}$ with P. Thus we add the suitable reduction rules:

$$\frac{}{\mathbb{D}(n[\,\mathbb{C}(\mathsf{in}\, m.P)\,]) \mid \mathbb{E}(m[\,R\,]) \to \mathbb{D}(\mathbf{0}) \mid \mathbb{E}(m[\,n[\,\mathbb{C}(P)\,] \mid R\,])} \text{ (In)}$$

$$\frac{}{m[\,\mathbb{D}(n[\,\mathbb{C}(\mathsf{out}\, m.P)\,])\,] \to n[\,\mathbb{C}(P)\,] \mid m[\,\mathbb{D}(\mathbf{0})\,]} \text{ (Out)}$$

$$\frac{}{\mathbb{C}(\mathsf{open}\, n.P) \mid \mathbb{D}(n[\,Q\,]) \to \mathbb{C}(P) \mid \mathbb{D}(\!|Q|\!)} \text{ (Open)} \qquad \frac{P \to Q}{(\!|P|\!) \to (\!|Q|\!)} \text{ (Brac)}$$

Structural Encoding. The encoding of a (possibly parenthesised) MA process is defined by delegating the management of an ambient $n[\,\cdot\,]$ to a suitable `link`-calculus process. The multi-party interaction between capabilities and ambients is regulated via communication on dedicated ports. Informally, a process P "resides" in some location, to which it addresses all its requests about the next actions to perform. The process that simulates the ambient $n[\,\cdot\,]$ also resides in some location $\tilde{p}$: it also defines an inner location $\tilde{a}$, where its content resides. A location $\tilde{a}$ denotes the 5-tuple of ports $a_{in}, a_{[in]}, a_{out}, a_{[out]}, a_{opn}$, where:

1. a_{in} is used for the interaction between the capability "$\mathsf{in}\, m$" to enter the ambient m and the ambient $n[\,\cdot\,]$ where it is contained;
2. $a_{[in]}$ for the interaction between the ambient $n[\,\cdot\,]$ that contains the capability "$\mathsf{in}\, m$" and the ambient $m[\,\cdot\,]$ to be entered;
3. a_{out} for the interaction between the capability "$\mathsf{out}\, m$" to exit the ambient m and the ambient $n[\,\cdot\,]$ where it is contained;
4. $a_{[out]}$ for the interaction between the ambient $n[\,\cdot\,]$ that contains the capability "$\mathsf{out}\, m$" and the ambient $m[\,\cdot\,]$ (that contains both) to be exited;
5. a_{opn} for the interaction between the capability "$\mathsf{open}\, n$" to open the ambient n and the sibling ambient $n[\,\cdot\,]$ to be dissolved.

In the encoding, dissolving ambients amounts to resorting to *forwarders*, located between the dissolved locations and the "parent" ones, that redirect all the interactions that involve the processes originally inside the dissolved ambient. Forwarders need to forward requests through arbitrarily long chains of indirection. These are requests arriving from "below" (i.e., from the processes inside the dissolved ambient) to be forwarded up and requests arriving from "above" (i.e., from the processes that want to interact with the processes inside the dissolved ambient) that must be forwarded down: the former case applies to all ports, whereas the second case applies only to ports $a_{[in]}$ and a_{opn}.

The encoding $[\![\, P \,]\!]_{\tilde{a}}$ of P is parametric with respect to its location, i.e., the tuple of ports to be used to communicate with the enclosing ambient (or forwarders). Name passing is used to match the name of the ambient for which the

$$
\begin{array}{lll}
[\![\,\mathbf{0}\,]\!]_{\tilde{a}} & \triangleq \mathbf{0} & [\![\,n[\,P\,]\,]\!]_{\tilde{a}} \triangleq (\nu\,\tilde{b})(Amb(n,\tilde{b},\tilde{a})|[\![\,P\,]\!]_{\tilde{b}}) \\
[\![\,P|Q\,]\!]_{\tilde{a}} & \triangleq [\![\,P\,]\!]_{\tilde{a}}|[\![\,Q\,]\!]_{\tilde{a}} & [\![\,\mathsf{in}\,m.P\,]\!]_{\tilde{a}} \triangleq {}^{\tau}\backslash_{a_{in}}\langle m,\underline{\tilde{x}}\rangle.[\![\,P\,]\!]_{\tilde{a}} \\
[\![\,(\nu\,n)P\,]\!]_{\tilde{a}} & \triangleq (\nu\,n)[\![\,P\,]\!]_{\tilde{a}} & [\![\,\mathsf{out}\,m.P\,]\!]_{\tilde{a}} \triangleq {}^{\tau}\backslash_{a_{out}}\langle m,\underline{\tilde{x}}\rangle.[\![\,P\,]\!]_{\tilde{a}} \\
[\![\,!P\,]\!]_{\tilde{a}} & \triangleq \mathsf{rec}X.\,([\![\,P\,]\!]_{\tilde{a}}|X) & [\![\,\mathsf{open}\,n.P\,]\!]_{\tilde{a}} \triangleq {}^{\tau}\backslash_{a_{opn}}\langle n\rangle.[\![\,P\,]\!]_{\tilde{a}} \\
 & & [\![\,(\!|P|\!)\,]\!]_{\tilde{a}} \triangleq (\nu\,\tilde{b})(Fwd(\tilde{b},\tilde{a})|[\![\,P\,]\!]_{\tilde{b}}
\end{array}
$$

$$
\begin{aligned}
Amb(n,\tilde{a},\tilde{p}) \triangleq\ & {}^{a_{in}}\backslash_{p_{[in]}}\langle\underline{m},\underline{\tilde{z}}\rangle.Amb(n,\tilde{a},\tilde{z}) + {}^{p_{[in]}}\backslash_{\tau}\langle n,\tilde{a}\rangle.Amb(n,\tilde{a},\tilde{p}) + \\
& {}^{a_{out}}\backslash_{p_{[out]}}\langle\underline{m},\underline{\tilde{z}}\rangle.Amb(n,\tilde{a},\tilde{z}) + {}^{a_{[out]}}\backslash_{\tau}\langle n,\tilde{p}\rangle.Amb(n,\tilde{a},\tilde{p}) + \\
& {}^{p_{opn}}\backslash_{\tau}\langle n\rangle.Fwd(\tilde{a},\tilde{p})
\end{aligned}
$$

$$
\begin{aligned}
Fwd(\tilde{a},\tilde{p}) \triangleq\ & {}^{a_{in}}\backslash_{p_{in}}\langle\underline{n},\underline{\tilde{x}}\rangle.Fwd(\tilde{a},\tilde{p}) + \\
& {}^{a_{[in]}}\backslash_{p_{[in]}}\langle\underline{n},\underline{\tilde{x}}\rangle.Fwd(\tilde{a},\tilde{p}) + {}^{p_{[in]}}\backslash_{a_{[in]}}\langle\underline{n},\underline{\tilde{x}}\rangle.Fwd(\tilde{a},\tilde{p}) + \\
& {}^{a_{out}}\backslash_{p_{out}}\langle\underline{n},\underline{\tilde{x}}\rangle.Fwd(\tilde{a},\tilde{p}) + {}^{a_{[out]}}\backslash_{p_{[out]}}\langle\underline{n},\underline{\tilde{x}}\rangle.Fwd(\tilde{a},\tilde{p}) + \\
& {}^{a_{opn}}\backslash_{p_{opn}}\langle\underline{n}\rangle.Fwd(\tilde{a},\tilde{p}) + {}^{p_{opn}}\backslash_{a_{opn}}\langle\underline{n}\rangle.Fwd(\tilde{a},\tilde{p})
\end{aligned}
$$

Fig. 4. Structural encoding of MA in `link`-calculus

capability is applicable with the name of the ambient where we would like to apply it. Moreover, in the case of enter/exit capabilities, name passing is necessary to inform the ambient that moves about the location where it is relocated.

We implicitly assume that all the restricted names introduced by the encoding are (globally) "fresh". The encoding is defined by straightforward structural induction in Fig. 4, where the processes $Amb(n,\tilde{a},\tilde{p})$ and $Fwd(\tilde{a},\tilde{p})$ represent, respectively, an ambient n located at $\tilde{p}$ and providing its content with location $\tilde{a}$, and a forwarder between the dissolved location $\tilde{a}$ and the "parent" location $\tilde{p}$.

As said above, applying the rule *(In)* to a process $n[\,\mathsf{in}\,m.P\,|\,Q\,]\,|\;\;m[\,R\,]$ requires at least a three-party interaction, whose corresponding encodings are commented below. To help intuition, we can represent ambients as graphs, that reflect the hierarchical structure of nested ambients and that record the locations of ambients reside and the inner locations where ambients contents reside. We now illustrate how the encoding works in Fig. 5, where the labels to be merged are in correspondence with the processes that issued them, and the processes are arranged according to the hierarchy of processes:

(i) The process $\mathsf{in}\,m.P$ with the capability to enter the ambient m is encoded by ${}^{\tau}\backslash_{a_{in}}\langle m,\underline{\tilde{x}}\rangle.[\![\,P\,]\!]_{\tilde{a}}$, where the emphasis is on the name m (variables in $\underline{\tilde{x}}$ are not important) and a_{in} is the port for the entering interaction.

(ii) The parent ambient $n[\,\cdot\,]$ to be moved is encoded by the restricted process $(\nu\,\tilde{a})(Amb(n,\tilde{a},\tilde{b})|[\![\,P\,]\!]_{\tilde{a}}|[\![\,Q\,]\!]_{\tilde{a}})$, where $Amb(n,\tilde{a},\tilde{b})$ includes the sub-process ${}^{a_{in}}\backslash_{b_{[in]}}\langle\underline{y},\underline{\tilde{z}}\rangle.Amb(n,\tilde{a},\tilde{z})$ as a choice.

(iii) The ambient $m[\,\cdot\,]$ to be entered is encoded by $(\nu\,\tilde{c})(Amb(m,\tilde{c},\tilde{b})|\;[\![\,R\,]\!]_{\tilde{c}})$, where $Amb(m,\tilde{c},\tilde{b})$ includes the process ${}^{b_{[in]}}\backslash_{\tau}\langle m,\tilde{c}\rangle.Amb(m,\tilde{c},\tilde{b})$ as a choice, where m must match with the first field of the message in the first item.

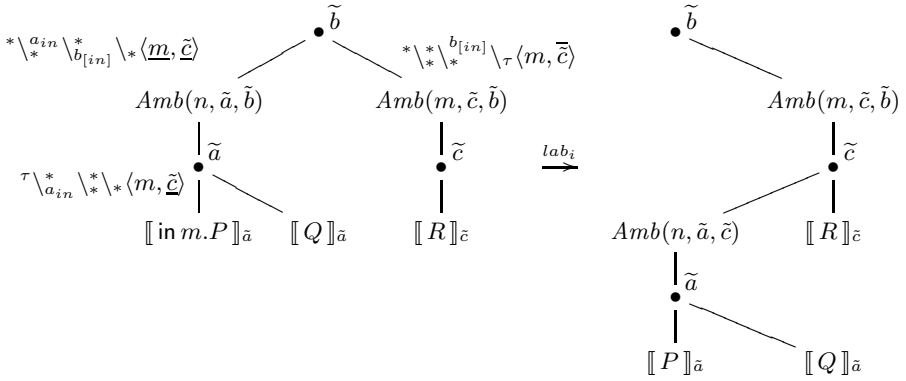

Fig. 5. Three party interaction for entering an ambient, where lab_i stands for $(\nu\,\tilde{a})(\nu\,\tilde{c})\,{}^{\tau}\backslash^{a_{in}}_{a_{in}}\backslash^{b_{[in]}}_{b_{[in]}}\backslash_{\tau} = {}^{\tau}\backslash^{\tau}_{\tau}\backslash^{b_{[in]}}_{b_{[in]}}\backslash_{\tau}$

The three process prefixes fit together: $\tilde{b}$ is the parent location of both ambients n and m, whose contents reside respectively at the sub-locations $\tilde{a}$ and $\tilde{c}$; the messages agree on the first value m (variable y is instantiated by m) and the links can be merged to form a valid link chain. Similarly, the variables $\tilde{x}$ and $\tilde{z}$ are instantiated by the location $\tilde{c}$. Intuitively, the complete interaction can be described as ${}^{\tau}\backslash^{a_{in}}_{a_{in}}\backslash^{b_{[in]}}_{b_{[in]}}\backslash_{\tau}\langle m,\tilde{c}\rangle$. However, names $\tilde{a}$ and $\tilde{c}$ are restricted, hence the link chain becomes ${}^{\tau}\backslash^{\tau}_{\tau}\backslash^{b_{[in]}}_{b_{[in]}}\backslash_{\tau}$ (by the application of the rule *(Res)* when $(\nu\,a)$ is encountered) and the tuple records the extrusion of $\tilde{c}$, i.e., it becomes $\langle m,\overline{\tilde{c}}\rangle$ (by the application of the rule *(Open)* when $(\nu\,c)$ is encountered). In the end, the whole transition is just labelled by ${}^{\tau}\backslash^{\tau}_{\tau}\backslash^{b_{[in]}}_{b_{[in]}}\backslash_{\tau}$, because the link chain is solid, and thus the (ground) tuple $\langle m,\overline{\tilde{c}}\rangle$ is discarded by rule *(Close)* (that also restores the restriction on the extruded location $\tilde{c}$ that was removed by the *(Open)* rule).

Similarly, as described in Fig. 6, the rule *(Out)* requires the encodings for: 1) the process $\mathsf{out}\,m.P$ capable to leave the ambient m; 2) its parent ambient $n[\,\cdot\,]$ to be moved; 3) the "grand-parent" ambient $m[\,\cdot\,]$ (enclosing $n[\,\cdot\,]$) to be exited.

Finally (see Fig. 6), in the rule *(Open)* besides the encodings for the processes $\mathsf{open}\,n.P$ and $n[\,Q\,]$, we should take care of the relocation of all processes included in the ambient n. We obtain this by using a forwarder, as explained below. The process with the open capability is encoded by ${}^{\tau}\backslash_{b_{opn}}\langle n\rangle.[\![\,P\,]\!]_{\tilde{b}}$, and the ambient to be dissolved by $(\nu\,\tilde{a})(Amb(n,\tilde{a},\tilde{b})|[\![\,P\,]\!]_{\tilde{a}})$, where $Amb(n,\tilde{a},\tilde{b})$ includes ${}^{b_{opn}}\backslash_{\tau}\langle n\rangle.Fwd(\tilde{a},\tilde{b})$ as a choice. The process $Fwd(\tilde{a},\tilde{b})$ presents a case for each kind of interaction, and it is used to suitably redirect all the interactions that involve the processes originally inside the dissolved ambient n. Trivially, rule *(Close)* can be applied, since ${}^{\tau}\backslash^{*}_{b_{opn}}\backslash_{*}\langle n\rangle \bullet {}^{*}\backslash^{b_{opn}}_{*}\backslash_{\tau}\langle n\rangle = {}^{\tau}\backslash^{b_{opn}}_{b_{opn}}\backslash_{\tau}\langle n\rangle$ that is a solid link chain (in this case there are no extruded names in the tuple).

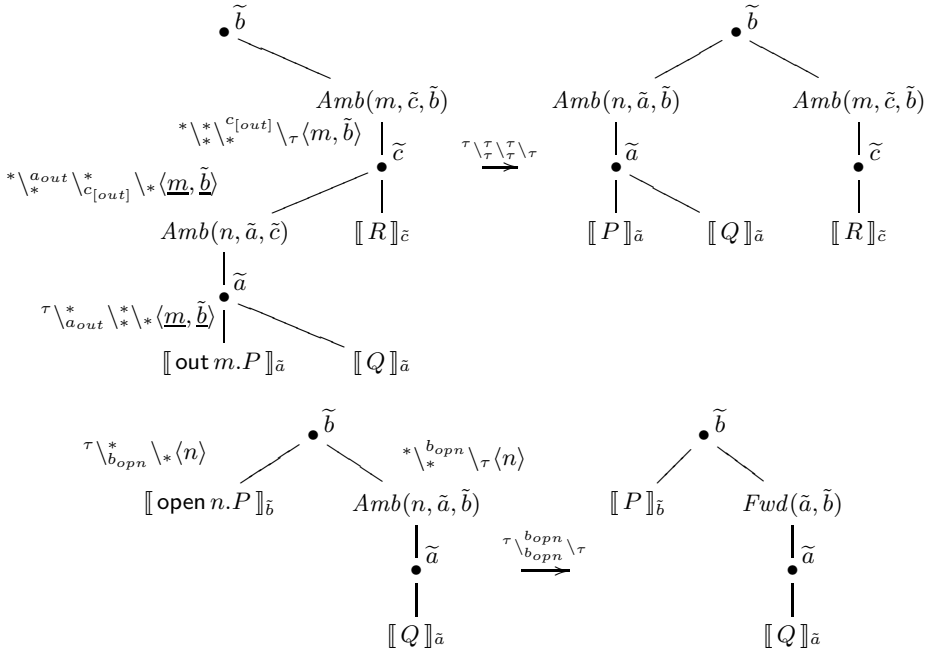

Fig. 6. (Top part) Three party interaction for exiting an ambient, (bottom part) three party interaction for opening an ambient

In the following, $\mathtt{t}$ stands for the topmost location in which the ambients reside.

Definition 5. *We let $\equiv_l$ be the structural congruence on* `link`*-calculus processes induced by the axioms:*

$P \mid \mathbf{0} \equiv_l P \qquad P \mid Q \equiv_l Q \mid P \qquad (P \mid Q) \mid R \equiv_l P \mid (Q \mid R)$
$(\nu\, n)\mathbf{0} \equiv_l \mathbf{0} \qquad (\nu\, n)(\nu\, m)P \equiv_l (\nu\, m)(\nu\, n)P$
$(\nu\, n)(P \mid Q) \equiv_l P \mid (\nu\, n)Q, \quad \textit{if} \;\; n \notin fn(P) \qquad recX.P \equiv_l P\{recX.P/P\}$

Lemma 8. *$P \equiv_l Q$ implies $P \sim_l Q$.*

Proposition 3. *The encoding in Fig. 4 is well-defined, in the sense that if $P \equiv Q$ then $[\![\, P \,]\!]_{\tilde{\mathtt{t}}} \equiv_l [\![\, Q \,]\!]_{\tilde{\mathtt{t}}}$.*

Without loss of generality, in the following we will consider `link`-calculus processes as taken up to the structural congruence $\equiv_l$.

Example 9. Take again the MA process $P \triangleq m[\, s[\,\mathsf{in}\, n.R\,] \mid T\,] \mid \mathsf{open}\, m.Q \mid n[\, G\,]$ in Ex. 8 for putting our encoding at work. Let

$$\begin{aligned} P_\pi \triangleq [\![\, P \,]\!]_{\tilde{\mathtt{t}}} = {} & [\![\, m[\, s[\,\mathsf{in}\, n.R\,] \mid T\,] \,]\!]_{\tilde{\mathtt{t}}} | [\![\,\mathsf{open}\, m.Q \,]\!]_{\tilde{\mathtt{t}}} | [\![\, n[\, G\,] \,]\!]_{\tilde{\mathtt{t}}} = \\ & (\nu\, \tilde{a})(Amb(m,\tilde{a},\tilde{\mathtt{t}}) | [\![\, s[\,\mathsf{in}\, n.R\,] \mid T \,]\!]_{\tilde{a}}) | [\![\,\mathsf{open}\, m.Q \,]\!]_{\tilde{\mathtt{t}}} | [\![\, n[\, G\,] \,]\!]_{\tilde{\mathtt{t}}} = \\ & (\nu\, \tilde{a})(Amb(m,\tilde{a},\tilde{\mathtt{t}}) | ([\![\, T \,]\!]_{\tilde{a}} | ((\nu\, \tilde{b})(Amb(s,\tilde{b},\tilde{a}) | {}^{\tau}\backslash_{b_{in}}\langle n,\underline{\tilde{x}}\rangle.[\![\, R \,]\!]_{\tilde{b}})) | \\ & {}^{\tau}\backslash_{\mathtt{t}_{opn}}\langle m\rangle.[\![\, Q \,]\!]_{\tilde{\mathtt{t}}} | (\nu\, \tilde{c})(Amb(n,\tilde{c},\tilde{\mathtt{t}}) | [\![\, G \,]\!]_{\tilde{c}}) \end{aligned}$$

be its encoding. Here we will recall the two transitions of process P, described in Ex. 8. The only difference is that the dissolved ambient is surrounded by $(\!|.|\!)$, in order to record the content of the opened ambient. The process P can execute an $\mathsf{open}\, n$ followed by a $\mathsf{in}\, n$ one:

$$P \equiv m[\, s[\, \mathsf{in}\, n.R\,] \mid T\,] \mid \mathsf{open}\, m.Q \mid n[\, G\,] \to (\!|s[\, \mathsf{in}\, n.R\,]|T|\!) \mid Q \mid n[\, G\,] \to (\!|T|\!) \mid Q \mid n[\, s[\, R\,]|G\,] \equiv P'$$

We will show that we can directly encode any derivatives on an MA process P obtaining a process which is congruent to a derivative of $[\![\, P\,]\!]_{\tilde{\mathtt{t}}}$. If we encode the target process $P' \equiv (\!|T|\!) \mid Q \mid n[s[R]|G]$ of the two MA transitions, we obtain:

$$\begin{aligned}
&[\![\, P'\,]\!]_{\tilde{\mathtt{t}}} = [\![\, (\!|T|\!)\,]\!]_{\tilde{\mathtt{t}}} | [\![\, Q\,]\!]_{\tilde{\mathtt{t}}} | [\![\, n[s[R]|G]\,]\!]_{\tilde{\mathtt{t}}} \\
&= (\nu\, \tilde{b})(Fwd(\tilde{b}, \tilde{\mathtt{t}}) | [\![\, T\,]\!]_{\tilde{b}}) | [\![\, Q\,]\!]_{\tilde{\mathtt{t}}} (\nu\, \tilde{c})(Amb(n, \tilde{c}, \tilde{\mathtt{t}}) | [\![\, G\,]\!]_{\tilde{c}} | (\nu\, \tilde{b})(Amb(s, \tilde{b}, \tilde{c}) | [\![\, R\,]\!]_{\tilde{b}})) \\
&\equiv_l (\nu\, \tilde{b})(\nu\, \tilde{c})(\nu\, \tilde{a})(Fwd(\tilde{b}, \tilde{\mathtt{t}}) | [\![\, T\,]\!]_{\tilde{b}} | [\![\, Q\,]\!]_{\tilde{\mathtt{t}}} | (Amb(n, \tilde{c}, \tilde{\mathtt{t}}) | [\![\, G\,]\!]_{\tilde{c}} | Amb(s, \tilde{b}, \tilde{c}) | [\![\, R\,]\!]_{\tilde{b}}) \quad (1)
\end{aligned}$$

where in the last step, we rearrange all the restrictions at the left end.
By following instead the derivatives of P_π, we have that the first transition, as illustrated below, corresponding to the "$\mathsf{open}\, m$" capability, derived by involving the two subprocesses $Amb(m, \tilde{a}, \tilde{\mathtt{t}}) | [\![\, T\,]\!]_{\tilde{a}}$ and ${}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.[\![\, Q\,]\!]_{\tilde{\mathtt{t}}}$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{{}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.Q_\pi \xrightarrow{{}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle} Q_\pi}\,(\text{Act}) \quad \dfrac{}{Amb(m, \tilde{a}, \tilde{\mathtt{t}}) \xrightarrow{{}^{\mathtt{t}_{opn}}\backslash_{\tau} \langle m \rangle} Fwd(\tilde{a}, \tilde{\mathtt{t}})}\,(\text{Act})}{{}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.Q_\pi | Amb(m, \tilde{a}, \tilde{\mathtt{t}}) \xrightarrow{{}^{\tau}\backslash^{\mathtt{t}_{opn}}_{\mathtt{t}_{opn}}\backslash_{\tau}} Q_\pi | Fwd(\tilde{a}, \tilde{\mathtt{t}})}\,(\text{Close})}{({}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.Q_\pi | Amb(m, \tilde{a}, \tilde{\mathtt{t}})) | T_\pi \xrightarrow{{}^{\tau}\backslash^{\mathtt{t}_{opn}}_{\mathtt{t}_{opn}}\backslash_{\tau}} Q_\pi | Fwd(\tilde{a}, \tilde{\mathtt{t}}) | T_\pi}\,(\text{Par})}{(\nu\, a)({}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.Q_\pi | Amb(m, \tilde{a}, \tilde{\mathtt{t}}) | T_\pi) \xrightarrow{{}^{\tau}\backslash^{\mathtt{t}_{opn}}_{\mathtt{t}_{opn}}\backslash_{\tau}} (\nu\, a)(Q_\pi | Fwd(\tilde{a}, \tilde{\mathtt{t}}) | T_\pi)}\,(\text{Res})}{(\nu\, a)({}^{\tau}\backslash_{\mathtt{t}_{opn}} \langle m \rangle.Q_\pi | Amb(m, \tilde{a}, \tilde{\mathtt{t}}) | T_\pi) | C_\pi \xrightarrow{{}^{\tau}\backslash^{\mathtt{t}_{opn}}_{\mathtt{t}_{opn}}\backslash_{\tau}} (\nu\, a)(Q_\pi | Fwd(\tilde{a}, \tilde{\mathtt{t}}) | T_\pi) | C_\pi}\,(\text{Par})$$

where $C_\pi \equiv (\nu\, \tilde{b})(Amb(s, \tilde{b}, \tilde{a}) | {}^{\tau}\backslash_{bin} \langle n, \underline{\tilde{x}} \rangle.[\![\, R\,]\!]_{\tilde{b}} | (\nu\, \tilde{c})(Amb(n, \tilde{c}, \tilde{\mathtt{t}}) \mid [\![\, G\,]\!]_{\tilde{c}})$, $Q_\pi \equiv [\![\, Q\,]\!]_{\tilde{\mathtt{t}}}$ and $T_\pi \equiv [\![\, T\,]\!]_{\tilde{a}}$.

The next transition (not reported for the sake of brevity) performs the corresponding "$\mathsf{in}\, n$" capability, leading to the target process:
$(\nu\, \tilde{a})(\nu\, \tilde{b})(\nu\, \tilde{c})([\![\, R\,]\!]_{\tilde{b}} | Amb(s, \tilde{b}, \tilde{c}) | Fwd(\tilde{a}, \tilde{\mathtt{t}}) | Amb(n, \tilde{c}, \tilde{\mathtt{t}}) | [\![\, T\,]\!]_{\tilde{a}} | [\![\, G\,]\!]_{\tilde{c}} | [\![\, Q\,]\!]_{\tilde{\mathtt{t}}})$ (2)
In our case also the $Fwd(\tilde{a}, \tilde{\mathtt{t}})$ process is involved, as the process $Amb(s, \tilde{b}, \tilde{a})$ refers to the ambient name $\tilde{a}$ that does not exist any longer. It is easy to see that (1) differs from (2) only for bound names and variables.

Operational Correspondence. We can filter out non solid transitions (representing partial interactions) of encoded processes by letting: $[\![\, P\,]\!] \triangleq (\nu\, \tilde{\mathtt{t}})[\![\, P\,]\!]_{\tilde{\mathtt{t}}}$

We say that a link chain s is *silent* if it consists of τ actions only, and write $P \xrightarrow{\tau} Q$ if $P \xrightarrow{s} Q$ for some silent s. (Note that any silent link chain is solid.)

Lemma 9. *If* $[\![\, P\,]\!] \xrightarrow{s} Q$, *then* s *is silent.*

Proof. The proof is by cases on the type of capability simulated by the encoded process $[\![P]\!]$ ($\mathsf{open}\, n.Q$, $\mathsf{in}\, n.Q$, $\mathsf{out}\, n.Q$). The proof of each case is, in turn, by induction on the length of the derivation of the transition.

Theorem 1. *Let P be a MA process, then $P \to P'$ if and only if there exists Q such that $[\![P]\!] \xrightarrow{s} Q$, and $Q \equiv [\![P']\!]$.*

The *only if* part can be proved by induction on the proof of reduction $P \to P'$. The idea underlying the proof of the *if* part goes as follows. At the extremities of the step a silent action τ should appear and the only actions able to perform τ on the left are the movement capabilities (offering the links ${}^{\tau}\backslash_{a_{in}}$, ${}^{\tau}\backslash_{a_{out}}$, ${}^{\tau}\backslash_{a_{opn}}$), while on the right we need an ambient offering the corresponding links ${}^{a'_{[in]}}\backslash_{\tau}$, ${}^{a'_{[out]}}\backslash_{\tau}$, ${}^{a'_{opn}}\backslash_{\tau}$. An ambient is needed to pass from the action a_{in} (resp. a_{out}) to the action $a'_{[in]}$ (resp. $a'_{[out]}$), while the forwarders do not change the type of actions. Participants to the action can be arranged in parallel, according to their position in s, thanks to the $\equiv_l$. By induction on the proof, we can rebuild the tree of the bracketed ambients involved in the reduction. Working modulo the structural congruence we can always build the synchronisation, starting from one of the extremities, thus rebuilding the reduction under analysis.

Linked bisimilarity induces a behavioural equivalence on MA processes via our encoding: two MA processes can be retained as equivalent if their encodings are so. We conjecture that linked bisimilarity is finer than barbed congruence, since it would distinguish e.g., $(\nu\, n)n[\mathsf{in}\, m.\mathbf{0}]$ from $\mathbf{0}$. Moreover, link bisimilarity has a 'strong' flavour (silent moves are matched exactly), as opposed to the 'weak' flavour of barbed congruence. We can define a weak version of linked bisimilarity in the standard way, but then the weak version would not be preserved by sum and still it would distinguish e.g., $(\nu\, n)n[\mathsf{in}\, m.\mathbf{0}]$ from $\mathbf{0}$. We think the mismatch is mostly due to the quite arbitrary choice of barbs to be observed in MA, i.e., the names of the topmost ambients: in our encoding, an ambient is just an interacting process, and its name is just a piece of information among others used to match capabilities requests.

5 Concluding Remarks and Related Works

We have presented the `link`-calculus as a lightweight enrichment of traditional dyadic process calculi able to deal with open multiparty interactions. We consider the `link`-calculus as a basis to investigate more general forms of interaction. An important field of application of our calculus is that of Systems Biology, where biological interactions are often multi-party. We would like to generalise link prefixes to *link-chain* prefixes, to encode some simple pattern of interaction directly, and also to express non linear communication patterns in the prefixes.

Among the recently presented network-aware extensions of classical calculi such as [14] (to handle explicit distribution, remote operations and process mobility), and [11] (to deal with permanent nodes crashing and links breaking), the closest proposal to ours is in [23], an extension of π-calculus, where links are named and are distinct from usual I/O actions, and there is one sender and

one receiver (the output includes the final receiver name). In our calculus, links can carry message tuples, and each participant can play both the sender and the receiver rôle. Our semantics recalls their concurrent semantics, where transmissions can be observed in the form of a multi-set of routing paths. In our case the collected links are organised in a link chain. In [6] the authors present a general framework to extend synchronisation algebras [27] with name mobility, that could be easily adapted to many other high-level kinds of synchronisation, like ours, but with a more complex machinery. More sophisticated forms of synchronisations, with a fixed number of processes, are introduced in π-calculus in [24] (joint input) and in [7] (polyadic synchronisation). The focus of [19] is instead on the expressiveness of an asynchronous CCS equipped with joint inputs allowing the interactions of n processes, proving that there is no truly distributed implementation of operators synchronising more than three processes. As in the join-calculus [13], and differently from our approach, participants can act either as senders or as receivers. In [17], a conservative extension of CCS, with multiparty synchronisation is introduced. The mechanism is realised as a sequence of dyadic synchronisations and, furthermore, puts some constraints that make the parallel operator non associative. Our approach also recalls the asynchronous semantics of CCS-like process calculi (see e.g., [26,10]). In both cases, the idea is that one process decides which interaction to try, and the other processes have to match. We introduce the capability of creating chains of links, useful to model communication patterns and information routing. Finally, [2] introduces a distributed version of the π-calculus for names to be exported, where names are equipped with the information needed to point back to its local environment, thus keeping track of the origin of mobile agents in a multi-hop travel.

Several approaches, among which we recall [12,20,25,3,4], are specifically devoted to provide an LTS semantics to MA, as a basis for bisimulation congruence. A contextual equivalence for MA is instead presented in [16]. While, in all these works *ad hoc* semantics are introduced, our proposed encoding is just an illustrative example of application of our network-aware calculus. Our link labels naturally accommodate the encoding. Furthermore, their LTSs are higher-order, since they can move processes (e.g., in [20], a transition can lead from a process to a context, while in [3] contexts are used as labels). A different approach is in [18], where a coalgebraic denotational semantics for the MA is presented.

References

1. Bodei, C., Brodo, L., Degano, P., Gao, H.: Detecting and preventing type flaws at static time. Journal of Computer Security 18(2), 229–264 (2010)
2. Bodei, C., Degano, P., Priami, C.: Names of the π-calculus agents handled locally. Theor. Comput. Sci. 253(2), 155–184 (2001)
3. Bonchi, F., Gadducci, F., Monreale, G.V.: Labelled transitions for mobile ambients (as synthesized via a graphical encoding). ENTCS 242(1), 73–98 (2009)
4. Bonchi, F., Gadducci, F., Monreale, G.V.: Reactive systems, barbed semantics, and the mobile ambients. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 272–287. Springer, Heidelberg (2009)

5. Brodo, L.: On the expressiveness of the π-calculus and the mobile ambients. In: Johnson, M., Pavlovic, D. (eds.) AMAST 2010. LNCS, vol. 6486, pp. 44–59. Springer, Heidelberg (2011)
6. Bruni, R., Lanese, I.: Parametric synchronizations in mobile nominal calculi. Theor. Comput. Sci. 402(2-3), 102–119 (2008)
7. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in pi-calculus. Nordic Journal of Computing 10(2), 70–98 (2003)
8. Cardelli, L.: Brane calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–278. Springer, Heidelberg (2005)
9. Cardelli, L., Gordon, A.D.: Mobile ambients. Theor. Comput. Sci. 240(1), 177–213 (2000)
10. de Boer, F.S., Palamidessi, C.: On the asynchronous nature of communication in concurrent logic languages: A fully abstract model based on sequences. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 99–114. Springer, Heidelberg (1990)
11. De Nicola, R., Gorla, D., Pugliese, R.: Basic observables for a calculus for global computing. Information and Computation 205(10), 1491–1525 (2007)
12. Ferrari, G.-L., Montanari, U., Tuosto, E.: A LTS semantics of ambients via graph synchronization with mobility. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) ICTCS 2001. LNCS, vol. 2202, pp. 1–16. Springer, Heidelberg (2001)
13. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proc. of POPL 1996, pp. 372–385. ACM Press (1996)
14. Francalanza, A., Hennessy, M.: A theory of system behaviour in the presence of node and link failure. Information and Computation 206(6), 711–759 (2008)
15. Gardner, P., Laneve, C., Wischik, L.: Linear forwarders. Inf. Comput. 205(10), 1526–1550 (2007)
16. Gordon, A., Cardelli, L.: Equational properties of mobile ambients. Math. Struct. in Comp. Sci. 13(3), 371–408 (2003)
17. Gorrieri, R., Versari, C.: An Operational Petri Net Semantics for A^2CCS. Fundam. Inform. 109(2), 135–160 (2011)
18. Hausmann, D., Mossakowski, T., Schröder, L.: A coalgebraic approach to the semantics of the ambient calculus. Theor. Comput. Sci. 366(1-2), 121–143 (2006)
19. Laneve, C., Vitale, A.: The expressive power of synchronizations. In: Proc. of LICS 2010, pp. 382–391. IEEE Computer Society (2010)
20. Merro, M., Nardelli, F.Z.: Behavioral theory for mobile ambients. Journal of the ACM 52(6), 961–1023 (2005)
21. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
22. Milner, R.: Communicating and mobile systems: the π-calculus. CUP (1999)
23. Montanari, U., Sammartino, M.: Network conscious pi-calculus: a concurrent semantics. ENTCS 286, 291–306 (2012)
24. Nestmann, U.: On the expressive power of joint input. ENTCS 16(2) (1998)
25. Rathke, J., Sobociński, P.: Deriving structural labelled transitions for mobile ambients. Information and Computation 208(10), 1221–1242 (2010)
26. Vaandrager, F.W.: On the relationship between process algebra and input/output automata. In: Proc. of LICS 1991, pp. 387–398. IEEE Computer Society (1991)
27. Winskel, G.: Synchronization trees. Theor. Comput. Sci. 34(1-2), 33–82 (1984)
28. Wischik, L., Gardner, P.: Explicit fusions. Theor. Comput. Sci. 340(3), 606–630 (2005)