RPO Semantics for Mobile Ambients

Filippo Bonchi, Fabio Gadducci, Valentina Monreale

Under consideration for publication in Math. Struct. in Comp. Science

RPO Semantics for Mobile Ambients

Filippo Bonchi, Fabio Gadducci, Giacoma Valentina Monreale

CNRS and LIP, ENS-Lyon

email address:*filippo.bonchi@ens-lyon.fr*

Dipartimento di Informatica, Università di Pisa

email address:{*vale, fabio*}*@di.unipi.it*

Received 2 March 2011

The paper focuses on the synthesis of labelled transition systems (LTSs) for process calculi, choosing as testbed Mobile Ambients (MAs). The proposal is based on a graphical encoding: a process is mapped into a graph equipped with interfaces, such that the denotation is fully abstract with respect to the standard structural congruence. Graphs with interfaces are amenable to the synthesis mechanism based on borrowed contexts (BCs), an instance of relative pushouts (RPOs). The BC mechanism allows the effective construction of a LTS that has graphs with interfaces as states and labels, and such that the associated bisimilarity is a congruence. Our paper focuses on the analysis of a LTS over processes as graphs with interfaces: we use the LTS on graphs to recover a LTS directly defined over the structure of MAs processes, further defining a set of SOS inference rules capturing the same operational semantics.

1. Introduction

Process calculi are a powerful formalism for the specification of concurrent and distributed systems. Their flexibility is proved by the wide range of different domains they found application in. Their expressiveness is guaranteed by the comparison with standard (e.g. functional) computational frameworks. Finally, their usability is supported by well-defined methodologies for the presentation of their semantics.

Concerning operational semantics, it is usually defined by means of labelled transitions systems (LTSs): a set of states of the system, plus a labelled relation describing its transitions. This easily allows for providing a behavioural semantics based on observations, looking at the labels of the evolutions that might be performed by the system.

Usually, these LTSs are obtained inductively by means of a set of inference rules. Nevertheless, to obtain such a compact presentation of a LTS is often complex, and relies on the ingenuity of the researcher, which is nowadays tested by the increasing complexity of the calculi. So, in particular after Milner's treatment of π-calculus (Mil99), it became customary to present the semantics of a calculus by a reduction semantics:

an unlabelled relation (also generated by a set of rules) modulo a structural congruence equating those processes which intuitively represent the same distributed system.

The considerations above can be applied *verbatim* to one of the most recent and fruitful proposal: the calculus of mobile ambients (CG00) (MAs). The analogy between ambients and network domains, addressed since the introduction of the calculus, and between ambients and molecular environments, often exploited in system biology, made MAs a centerpiece in recent applications and developments of the process calculi paradigm.

It is then baffling that the calculus has been so resilient to the introduction of an observational semantics. This is likely due to the fact that already the set of rules defining the original reduction semantics of MAs is rather complex. As an example, the system evolution stating the "exporting" of a process P out of an ambient named n is represented by the rule $m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$. The rule needs to carry around the occurrences of processes Q and R, which denote the context into which the instance of the rule has to be mapped into. The need of such a rich contextual information makes difficult to obtain a satisfying observational semantics. After the early proposals by Cardelli and Gordon (GC03), and by Ferrari, Montanari and Tuosto (FMT01), we are only aware of the work by Merro and Zappa-Nardelli (MZN05) and, quite recently, by Rathke and Sobociński, originally presented in (RS08b) and partly modified in (RS10).

With the exception of (RS08b; RS10), the other proposals are not based on a methodological assumption. Indeed, only a recent series of papers addressed the problem of automatically synthesizing a LTS out of the reduction semantics of a calculus, further guaranteeing that the derived observational semantics is a congruence. The most successful technique so far was proposed by Leifer and Milner, based on the notion of *relative pushout* (RPO) (LM00): it captures in an abstract setting the intuitive notion of minimal context into which a process has to be inserted, in order to enable a reduction to occur.

However, proving that a calculus satisfies the requirements needed for applying the RPOs technique is often a daunting task. A way out of the impasse is represented by the graphical encodings of processes, turning structural congruence into graph isomorphism. Graph formalisms are usually amenable to the RPOs technique, and once the processes of a calculus have been encoded as graphs, a LTS can be distilled. Indeed, it seems no chance that the main source of examples concerning RPOs have been *bigraphs* (Mil06).

It is noteworthy that, should the reduction relation over graphs be defined using the double pushout (DPO) approach (BCE$^+$99), these graphs are amenable to the *borrowed context* (BC) technique (EK06), developed by Ehrig and König, which offers an algorithmic solution for calculating the minimal contexts enabling a graph transformation rule. Indeed, graphs form an *adhesive* category (LS05) hence, via the so-called *cospan* construction (SS05), BCs and RPOs are proved to be coincident notions (Sob04).

So, the approach pursued in this and other papers (BGK06; GM05) is quite straightforward: for a given calculus, a graphical encoding is found such that structural congruence is preserved, and the reduction semantics is captured by a set of graph transformation rules, specified using the DPO approach. A LTS for the calculus can thus be distilled. This is the way which allowed to derive the unique successful application so far of the RPO technique to the set of recursive processes of a calculus, still recovering the standard bisimulation congruence, even if for one of the simplest calculi, Milner's CCS (Mil89).

This paper exploits the graphical encoding for MAs proposed in (GM08) in order to distill a LTS on (processes encoded as) graphs. This LTS is used to infer a set of SOS rules defined on the processes of MAs, and to compare it with the alternative solution proposed in (RS08b) (also inspired by the RPO technique), discovering many similarities yet with a few substantial differences, as summed up in the concluding section.

Synopsis Section 2 summarizes a few notions concerning MAs. Section 3 introduces typed (hyper-)graphs with interfaces, while Section 4 recalls the DPO approach to graph rewriting and the associated BC technique for distilling a LTS. Section 5 discusses a graphical encoding for MAs processes, while Section 6 states the simulation of process reduction by DPO rewriting. The simulation is needed in Section 7 for the presentation of a LTS for graphs with interfaces representing MAs processes, which is exploited in Section 8 to introduce a LTS defined directly over MAs processes. Section 9 then presents a novel description of the distilled LTS by means of a set of SOS-style inference rules, which is finally used in Section 10 to prove the correspondence between our proposal and Rathke and Sobociński's. Section 11 concludes the paper.

Previous works This work builds on (BGM09a): the presentation of the single sections has been tightened and reshaped, inserting detailed proofs and a few additional examples. Two novel sections, Section 9 and Section 10, are partly drawn (with heavy restructuring) from (BGM09c). The SOS presentation of the distilled LTS is a fundamental feature, in order to reason about the structural properties of the operational semantics: its presence here strengthen our claims of the practical relevance of our proposal (Section 9). The coincidence between our LTS and Rathke and Sobociński's further witnesses the soundness of the operational semantics for MAs that we propose (Section 10).

2. Mobile Ambients

This section briefly recalls the finite, communication-free fragment of mobile ambients (CG00), its structural equivalence and reduction semantics. Note that, for the need of the presentation of the operational semantics in Section 8, we introduce an extended syntax that allows us to build processes containing *process variables* and *name variables*.

Definition 2.1 (Extended processes). Let $\mathcal{N}$ be a set of *names* ranged over by $m, n, u, \ldots$, $\mathcal{X} = \{X, Y, \ldots\}$ a set of *process variables* and $\mathcal{V} = \{x, y, \ldots\}$ a set of *name variables*. An *extended process* is a term generated by the syntax in Table 1.

Intuitively, an extended process such as $x[P]|X$ represents an underspecified process, where either the process X or the name of the ambient $x[-]$ can be further instantiated.

$P ::= 0, n[P], M.P, (\nu n)P, P_1\|P_2, X, x[P]$	$M ::= in\ n, out\ n, open\ n$

Table 1. Extended syntax of mobile ambients.

$P\|Q \equiv Q\|P$	$(P\|Q)\|R \equiv P\|(Q\|R)$
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	$(\nu n)(P\|Q) \equiv P\|(\nu n)Q$ if $n \notin fn(P)$
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	$(\nu n)m[P] \equiv m[(\nu n)P]$ if $n \neq m$
$P \equiv Q \Rightarrow M.P \equiv M.Q$	$P\|\mathsf{0} \equiv P$
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	$(\nu n)M.P \equiv M.(\nu n)P$ if $n \notin fn(M)$
$P \equiv Q \Rightarrow P\|R \equiv Q\|R$	$(\nu n)P \equiv (\nu m)(P\{^m/_n\})$ if $m \notin fn(P)$

Table 2. Structural congruence on pure processes.

Definition 2.2 (Pure and linear processes). An extended process is a *pure* process if no process or name variable occurs in it; it is a *linear* process if no process or name variable occurs in it more than once.

We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of pure processes; and $P_\epsilon, Q_\epsilon, R_\epsilon, \ldots$ over the set $\mathcal{P}_\epsilon$ of linear processes. We use the standard definitions for the set of free names of a pure process P, denoted $fn(P)$, and for α-convertibility, with respect to the restriction operators. Variables carry no name, hence $fn(x[P_\epsilon]) = fn(P_\epsilon)$ and $fn(X) = \emptyset$; the sets of name and process variables are defined as expected and denoted $nv(P_\epsilon)$ and $pv(P_\epsilon)$.

Moreover, we consider a family of *substitutions*, which may replace a process/name variable with a pure process/name, respectively. Substitutions avoid name capture: for a pure process P, the expression $(\nu n)(\nu m)(m[X]|x[\mathsf{0}])\{^m/_x, ^{n[P]}/_X\}$ corresponds to the pure process $(\nu p)(\nu q)(q[n[P]]|m[\mathsf{0}])$, for names $p, q \notin \{m\} \cup \mathtt{fn}(n[P])$.

The semantics exploits a *structural congruence*, denoted $\equiv$, which is the least equivalence on pure processes that satisfies the equations in Table 2. The congruence relates processes which intuitively specify the same system, up-to a syntactical rearrangement of its components, and it is then used to define a *reduction relation*, introduced below and denoted $\rightarrow$. The relation describes the evolution of processes over time: $P \rightarrow Q$ means that P reduces to Q, i.e., P executes a computational step and evolves into Q.

Definition 2.3 (Reduction relation). The *reduction relation* $\mathcal{R}_{amb} \subseteq \mathcal{P} \times \mathcal{P}$ is the smallest relation, closed under $\equiv$, generated by the rules in Table 3.

Our chosen congruence slightly differs from the standard one: we drop $(\nu n)\mathsf{0} \equiv \mathsf{0}$ and we add $(\nu n)M.P \equiv M.(\nu n)P$, allowing a restriction to enter a capability. The reduction semantics does not substantially change: the equality induced by the latter axiom holds in all the behavioural equivalences for mobile ambients that we are aware of. In particular, two processes that are structurally congruent are *reduction barbed* congruent (MZN05).

Example 2.1. As a running example, consider processes $Q = n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}]$ and $P = (\nu n)Q$. The application of the (InRed) axiom to Q results in $m[n[\mathsf{0}]|out\ m.\mathsf{0}]$, hence we may apply the rule (ResRed) to P and conclude $P \rightarrow (\nu n)(m[n[\mathsf{0}]|out\ m.\mathsf{0}])$.

(InRed)	$n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	(ResRed)	$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
(OutRed)	$m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	(AmbRed)	$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$
(OpenRed)	$open\ n.P\|n[Q] \rightarrow P\|Q$	(ParRed)	$P \rightarrow Q \Rightarrow P\|R \rightarrow Q\|R$

Table 3. Reduction relation on pure processes.

3. Graphs and Their Extension with Interfaces

We recall a few definitions concerning typed (hyper-)graphs, and their extension with *interfaces*, referring e.g. to (CG99) for a more detailed introduction.

Definition 3.1 (Typed graphs). A *(hyper-)graph* is a four-tuple $\langle N, E, s, t\rangle$, for N, E the sets of nodes and (hyper-)edges and $s, t : E \to N^*$ the source and target functions. A *graph morphism* is a pair of functions $\langle f_N, f_E\rangle$ preserving source and target.

Let T be a graph. A *typed graph* G over T is a pair $\langle |G|, t_G\rangle$, for $|G|$ a graph and $t_G : |G| \to T$ the typing morphism. A *T-typed graph morphism* is a graph morphism $f : |G_1| \to |G_2|$ between the underlying graphs preserving the typing.

The category of graphs typed over T is denoted by T-**Graph**.

Definition 3.2 (Graphs with interfaces). Let J and K be typed graphs. A *graph with input interface J and output interface K* is a triple $\mathbb{G} = \langle j, G, k\rangle$, for G a typed graph and $j : J \to G$, $k : K \to G$ the *input* and *output* morphisms. Let $\mathbb{G}$ and $\mathbb{H}$ be graphs with the same interfaces. An *interface graph morphism* is a typed graph morphism $f : G \to H$ between the underlying typed graphs preserving the input and output.

We let $J \xrightarrow{j} G \xleftarrow{k} K$ denote a graph with interfaces J and K. If the interfaces J, K are *discrete*, i.e., they contain only nodes, we represent them by sets; if K is the empty set, we often denote a graph with interfaces as a graph morphism $J \to G$.[†]

In order to define the encoding, some operators on graphs with discrete interfaces are needed. Since we rely on the proposal in (GM08), we refer the reader there for details.

Definition 3.3 (Two operators). Let $\mathbb{G} = I \xrightarrow{j} G \xleftarrow{k} K$ and $\mathbb{G}' = K \xrightarrow{j'} G' \xleftarrow{k'} J$ be graphs with discrete interfaces. Their *sequential composition* is the graph with discrete interfaces $\mathbb{G} \circ \mathbb{G}' = I \xrightarrow{j''} G'' \xleftarrow{k''} J$, for G'' the disjoint union $G \uplus G'$, modulo the equivalence induced by $k(x) = j'(x)$ for all $x \in N_K$, and j'', k'' the uniquely induced arrows.

Let $\mathbb{G} = J \xrightarrow{j} G \xleftarrow{k} K$ and $\mathbb{G}' = J' \xrightarrow{j'} G' \xleftarrow{k'} K'$ be graphs with discrete, compatible interfaces.[‡] Their *parallel composition* is the graph with discrete interfaces $\mathbb{G} \otimes \mathbb{G}' = (J \cup J') \xrightarrow{j''} G'' \xleftarrow{k''} (K \cup K')$, for G'' the disjoint union $G \uplus G'$, modulo the equivalence induced by $j(x) = j'(x)$ for all $x \in N_J \cap N_{J'}$ and $k(y) = k'(y)$ for all $y \in N_K \cap N_{K'}$, and j'', k'' the uniquely induced arrows.

The sequential composition $\mathbb{G} \circ \mathbb{G}'$ take the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{G}'$, gluing the outputs of $\mathbb{G}$ with the corresponding inputs of $\mathbb{G}'$. The parallel composition $\mathbb{G} \otimes \mathbb{G}'$ takes the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{G}'$, and glue the inputs (outputs) of $\mathbb{G}$ with the corresponding inputs (outputs) of $\mathbb{G}'$. Both operations are defined on "concrete" graphs, even if the result is independent of the chosen representatives.

[†] We often refer implicitly to a graph with interfaces as the representative of its isomorphism class, using the same symbols to denote it and its components. Moreover, in the discrete iterfaces case, we refer to the nodes in the image of the input and output morphisms as inputs and outputs, respectively.

[‡] That is, any node in $N_J \cap N_{J'}$ has the same type in J and J' (and similarly for $N_K \cap N_{K'}$).

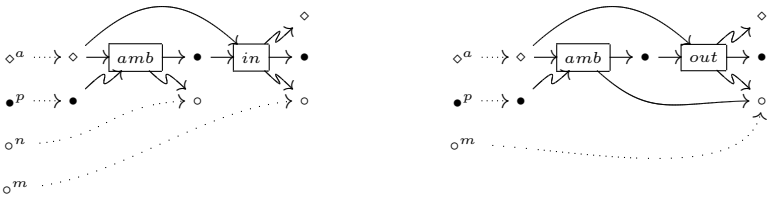


Fig. 1. Graphs with interfaces $\mathbb{G}$ and $\mathbb{G}'$ (from left to right).

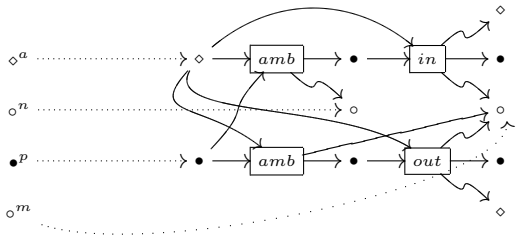


Fig. 2. Graph with interfaces $\mathbb{G} \otimes \mathbb{G}'$.

Example 3.1. Let us consider the two graphs with interfaces $\mathbb{G} = \{a, p, n, m\} \to G \leftarrow \emptyset$ and $\mathbb{G}' = \{a, p, m\} \to G' \leftarrow \emptyset$ in Fig. 1. As it is going to be explained in Section 5, these two graphs respectively represent the graphical encodings of the MAs processes $Q_1 = n[in\ m.\mathsf{0}]$ and $Q_2 = m[out\ m.\mathsf{0}]$. For the moment, the reader can ignore how these encodings are obtained. We only observe that in the graph $\mathbb{G}$ there is an edge *amb* representing the ambient n and an edge *in* simulating the capability *in* m. Analogously, in the graph $\mathbb{G}'$ there is an edge *amb* representing the ambient m and an edge *out* simulating the capability *out* m. Moreover, ambient names are represented by nodes of type $\circ$ that are in the input interfaces of the two graphs, and processes (subprocess) are represented by graphs (subgraphs) that have as root a pair of nodes $\langle \bullet, \diamond \rangle$. Only the root nodes $\langle \bullet, \diamond \rangle$ of the graphs representing the processes Q_1 and Q_2 are in the input interfaces of the corresponding graphs. Moreover, as we can note, each subprocess is represented by a subgraph that has a different $\bullet$ root node, while sometimes subgraphs representing different subterms share the $\diamond$ root node. We shall see later on why this occurs.

The two graphs $\mathbb{G}$ and $\mathbb{G}'$ have compatible interfaces. Indeed, it is easy to check that the type of the nodes belonging to both input interfaces coincides. Therefore, it is possible to compute the parallel composition of the graphs $\mathbb{G}$ and $\mathbb{G}'$, resulting in the graph with interfaces shown in Fig. 2. It is easy to note that it is obtained by making the union of the input interfaces and of the output interfaces, respectively, and the disjoint union of G and G', gluing the root nodes of both graphs and the nodes representing the name m. As explained later in Section 5, the graph with interfaces obtained by the parallel composition of $\mathbb{G}$ and $\mathbb{G}'$ represents the process obtained by making the parallel composition between Q_1 and Q_2, that is, the process $Q_1|Q_2$.

4. On Graphs with Interfaces and Borrowed Contexts

This section introduces the *double-pushout* (DPO) approach to the rewriting of graphs with interfaces and its extension with *borrowed contexts* (BCs), as introduced in (EK06).

Definition 4.1 (Graph production). A T-typed *graph production* is a span of morphisms $L \overset{l}{\hookleftarrow} I \overset{r}{\longrightarrow} R$ in T-**Graph**, such that l is mono. A T-typed *graph transformation system* (GTS) $\mathcal{G}$ is a pair $\langle P, \pi \rangle$ where P is a set of production names and π is a function assigns a T-typed production to each production name.

A production identifies those graph items that should be rewritten, and specifies how new items have to be inserted. The intuitive meaning is captured by the definition below.

Definition 4.2 (Direct derivation). Let $J \to G$ and $J \to H$ be two graphs with interfaces and $p\colon L \hookleftarrow I \to R$ a production. A *match* of p in G is a morphism $m\colon L \to G$. A *direct derivation* from $J \to G$ to $J \to H$ via p and m is a commuting diagram as in Fig. 3, such that two squares are pushouts (PO), denoted $J \to G \Longrightarrow J \to H$.

The morphism $k : J \to C$ (making the left triangle commute) is unique, whenever it exists. If such a morphism does not exist, the rewriting step is not feasible.

In these derivations, the left-hand side L of a production must then occur completely in G. In a *borrowed context* (BC) derivation L might occur partially in G, since the latter may interact with the environment through the interface J in order to exactly match L. Those BCs are the "smallest" contexts needed to obtain the image of L in G, and they may be used as suitable labels.

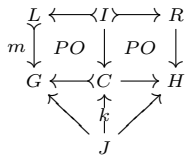


Fig. 3. A direct derivation.

Definition 4.3 (Rewriting with Borrowed Contexts). Given a production $p\colon L \hookleftarrow I \to R$, a graph with interfaces $J \to G$ and a span of monos $d : G \hookleftarrow D \rightarrowtail L$, we say that $J \to G$ *reduces to* $K \to H$ with label $J \rightarrowtail F \hookleftarrow K$ *via* p and d if there are graphs G^+, C and additional morphisms such that the diagram in Fig. 4 commutes and the squares are either pushouts (PO) or pullbacks (PB). We write $J \to G \overset{J \rightarrowtail F \hookleftarrow K}{\longrightarrow} K \to H$, called *rewriting step with borrowed context*.

The upper left-hand square of the diagram in Fig. 3 merges the left-hand side L and the graph G to be rewritten according to a partial match $G \hookleftarrow D \rightarrowtail L$. The resulting G^+ contains a total match of L and is rewritten as in the DPO approach, producing the two other squares in the upper row. The pushout in the lower row gives the BC F which is missing for obtaining a total match of L, along with a morphism $J \rightarrowtail F$ indicating how F should be pasted to G. Finally, the interface for H is obtained by "intersecting" F and C via a pullback.

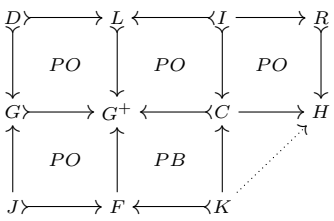


Fig. 4. A BC derivation.

Note that two pushout complements that are needed in Definition 4.3, namely C and F, may not exist. In this case, the rewriting step is not feasible.

5. Graphical Encoding for Processes

This section shortly recalls a graphical encoding for MAs processes presented (with minor variants) in (GM08). After the description of a type graph (T_M, in Fig. 5), the encoding is defined inductively by means of the composition operators introduced in Definition 3.3. This roughly corresponds to the standard construction of the tree for a term of an algebra: names are interpreted as variables, so they are mapped to graph leaves and can be shared.

Intuitively, a node of type $\circ$ represents an ambient name, while a graph that has as roots a pair of nodes $\langle \diamond, \bullet \rangle$ represents a process. More precisely, the node of type $\diamond$ represents the activating point for reductions of the process represented by the graph. We need two different types of node to model processes by graphs, because each graph has to model both syntactical and activation dependencies between the operators of a process.

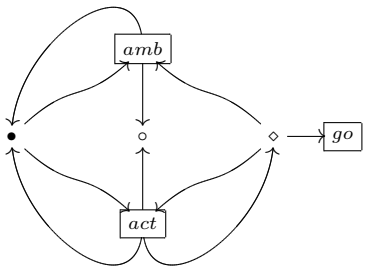


Fig. 5. The type graph T_M (for $act \in \{in, out, open\}$).

Each edge of T_M, except the *go* edge, simulates an operator of MAs. Note that the *act* edge stands for three edges, namely *in*, *out* and *open*. These edges simulate the capabilities of the calculus, while the *amb* edge simulates the ambient operator, and no edge simulates either the restriction or the parallel composition. The *go* edge is a syntactical device for detecting the "entry" point for the computation, needed to simulate MAs reduction semantics: it allows to avoid the occurrence of a reduction underneath an *act* operator.

Figs. 6, 7 and 8 depict a class of graphs: linear processes are encoded into an expression containing those graphs as constants, and parallel and sequential composition as operators. We assume a set $\{a, p\} \uplus \{X_a, X_p \mid X \in \mathcal{X}\}$ with no intersection with $\mathcal{N}$.

We use $\mathtt{0}_{a,p}$ and $id_{a,p}$ as shorthands for $\mathtt{0}_a \otimes \mathtt{0}_p$ and $id_a \otimes id_p$, respectively; and similarly $\mathtt{0}_X$ and id_X stand for $\mathtt{0}_{X_a} \otimes \mathtt{0}_{X_p}$ and $id_{X_a} \otimes id_{X_p}$. Moreover, for a set of names Γ, we use $\mathtt{0}_\Gamma$ and id_Γ as shorthands for $\bigotimes_{n \in \Gamma} \mathtt{0}_n$ and $\bigotimes_{n \in \Gamma} id_n$, respectively; and for a process P_ϵ, $id_{pv(P_\epsilon)}$ stands for $\bigotimes_{X \in pv(P_\epsilon)} id_X$. The preceding expressions are well-defined, because the $\otimes$ operator is associative. The definition below introduces the encoding of extended processes (with no occurrence of name variables) into graphs with interfaces, mapping a process into a graph expression. $[\![M.P_\epsilon]\!]_\Gamma$ denotes the encoding of *in* $n.P_\epsilon$, *out* $n.P_\epsilon$ and *open* $n.P_\epsilon$, while act_n represents the in_n, out_n and $open_n$ graphs, respectively.

Definition 5.1 (Encoding for processes). Let P_ϵ be a linear process with no occurrence of name variables and let Γ be a set of names such that $fn(P_\epsilon) \subseteq \Gamma$. The *encoding* of P_ϵ, denoted by $[\![P_\epsilon]\!]_\Gamma$, is defined by structural induction according to the rules in Table 4.

Given a linear process P and a set of names Γ, such that $fn(P_\epsilon) \subseteq \Gamma$, its encoding $[\![P_\epsilon]\!]_\Gamma$ is a graph with interfaces $(\{a, p\} \uplus \{X_a, X_p \mid X \in pv(P_\epsilon)\} \uplus \Gamma, \emptyset)$. Its *enriched encoding* is the graph $[\![P_\epsilon]\!]_\Gamma \otimes go$, denoted $[\![P_\epsilon]\!]^{go}_\Gamma$, and intuitively links a *go* edge to the $\diamond$ root node of each graph representing a process. This edge is needed for detecting the 'entry' point for the computation of the process. Its usee will become clearer later on.

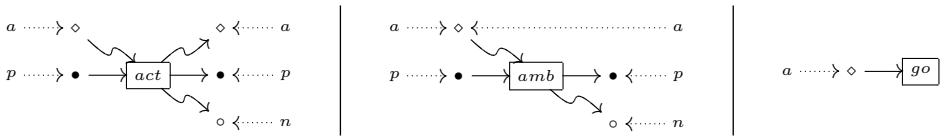


Fig. 6. Graphs act_n (with $act \in \{in, out, open\}$); amb_n; and go (left to right).

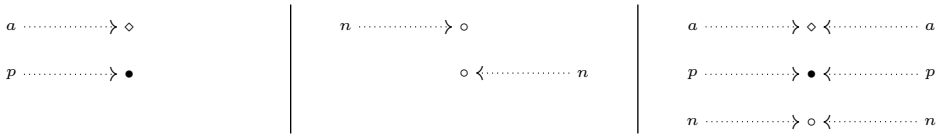


Fig. 7. Graphs $\mathtt{0}_a$ and $\mathtt{0}_p$; $\mathtt{0}_n$ and new_n; id_a, id_p and id_n (top-down, left to right).

Fig. 8. Graphs $\mathtt{0}_{X_a}$ and $\mathtt{0}_{X_p}$; id_{X_a} and id_{X_p} (top-down, left to right).

The encoding is fully abstract with respect to structural congruence (GM08, Thm 1).

Proposition 5.1. Let P, Q be pure processes and let Γ be a set of names, such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then, $P \equiv Q$ if and only if $[\![P]\!]^{go}_{\Gamma} = [\![Q]\!]^{go}_{\Gamma}$.

The result could be suitably extended, in order to encompass also linear processes.

Example 5.1. Consider the pure process $Q = n[in\ m.\mathtt{0}]|m[out\ m.\mathtt{0}]$ from Example 2.1. Fig. 2 shows the encoding $[\![Q]\!]_{\{n,m\}}$, as for Definition 5.1. The upper sub-graph in the encoding represents the left parallel component of Q, the process $n[in\ m.\mathtt{0}]$; the lower sub-graph represents the right component of the parallel operator, $m[out\ m.\mathtt{0}]$. The sub-graphs have the same roots, into which the nodes of the interface a and p are mapped, and they share the name node representing the ambient name m, which is used in both terms. This happens because the encoding is obtained via the $\otimes$-composition (Definition 3.3) of $[\![n[in\ m.\mathtt{0}]]\!]_{\{n,m\}}$ and $[\![m[out\ m.\mathtt{0}]]\!]_{\{n,m\}}$, shown in Fig. 1 (left to right).

The enriched encoding $[\![P]\!]^{go}_{\{m\}}$ for pure process $P = (\nu n)Q$, also from Example 2.1, is in Fig. 9. It is obtained from $[\![Q]\!]_{\{n,m\}}$ in two steps: the node n is first removed from the interface (getting $[\![P]\!]_{\{m\}}$); and the go edge is then attached to the activation node $\diamond$.

Let us focus on the first step: by definition, $[\![P]\!]_{\{m\}} = (new_n \otimes id_m \otimes id_{a,p}) \circ [\![Q]\!]_{\{n,m\}}$. The graph with interface $(new_n \otimes id_m \otimes id_{a,p})$ has the same underlying graph of $id_{n,m,a,p}$, but name n is missing from the input interface: its sequential composition with a graph having interface $\{n, m, a, p\}$ results into the same graph but without n among its inputs.

As for the second step, note that the activation node $\diamond$ is linked to the edges representing the ambients n and m and to the capabilities $in\ m$ and $out\ m$: attaching the go edge to $\diamond$ will allow the ambients connected to it to be involved into a reduction.

$[\![X]\!]_\Gamma$	=	$\mathtt{0}_X \otimes \mathtt{0}_\Gamma$	
$[\![\mathtt{0}]\!]_\Gamma$	=	$\mathtt{0}_{a,p} \otimes \mathtt{0}_\Gamma$	
$[\![n[P_\epsilon]]\!]_\Gamma$	=	$(id_{pv(P_\epsilon)} \otimes amb_n \otimes id_\Gamma) \circ [\![P_\epsilon]\!]_\Gamma$	
$[\![M.P_\epsilon]\!]_\Gamma$	=	$(id_{pv(P_\epsilon)} \otimes act_n \otimes id_\Gamma) \circ [\![P_\epsilon]\!]_\Gamma$	
$[\![P_\epsilon \mid Q_\epsilon]\!]_\Gamma$	=	$[\![P_\epsilon]\!]_\Gamma \otimes [\![Q_\epsilon]\!]_\Gamma$	
$[\![(\nu n)P_\epsilon]\!]_\Gamma$	=	$(id_{pv(P_\epsilon)} \otimes id_{a,p} \otimes new_m \otimes id_\Gamma) \circ [\![P_\epsilon\{^m/_n\}]\!]_{\Gamma\cup\{m\}}$	for $m \notin \Gamma$

Table 4. Encoding for pure processes.

6. Graph Transformation for Mobile Ambients

This section presents a GTS that models the reduction semantics of MAs. Fig. 10 presents the rules of the GTS $\mathcal{R}_{amb}$, which simulates the reduction semantics $\rightarrow$ introduced in Section 2. The GTS $\mathcal{R}_{amb}$ contains just three rules: p_{in}, p_{out}, and p_{open}. They simulate the three axioms of the reductions relation: the rule p_{open} stands for the (OpenRed) axiom, and the same occurs for the others. Moreover, since we consider injective matches, we need an instance for the rules p_{in} and p_{out}, where the nodes labelled n and m may actually be coalesced, denoted p_{in-c} and p_{out-c}, respectively (not presented here).

Node identifiers describe the rules action, and they are of course arbitrary: they correspond to the elements of the set of nodes and characterize the span of functions.

Three (plus two) rules suffice for recasting the reduction semantics of mobile ambients. Indeed, the closure of reduction with respect to contexts is obtained by the embedding of a graph within a larger one. And no rule instance is needed: graph isomorphism takes care of structural congruence, and interfaces of the renaming of free names.

Our encoding is fully abstract with respect to the reduction relation $\rightarrow$ (GM08, Thm 2).

Theorem 6.1 (Reductions vs. rewrites). Let P be a pure process, and let Γ be a set of names, such that $fn(P) \subseteq \Gamma$. If $P \rightarrow Q$, then $\mathcal{R}_{amb}$ entails a direct derivation $[\![P]\!]^{go}_\Gamma \Longrightarrow [\![Q]\!]^{go}_\Gamma$. Vive versa, if $\mathcal{R}_{amb}$ entails a direct derivation $[\![P]\!]^{go}_\Gamma \Longrightarrow \mathbb{G}$, then there exists a pure process Q, such that $P \rightarrow Q$ and $\mathbb{G} = [\![Q]\!]^{go}_\Gamma$.

The correspondence holds since a rule is applied only if there is a match that covers a sub-graph with the *go* operator on top. This allows the occurrence of reductions inside activated ambients, but not inside capabilities.

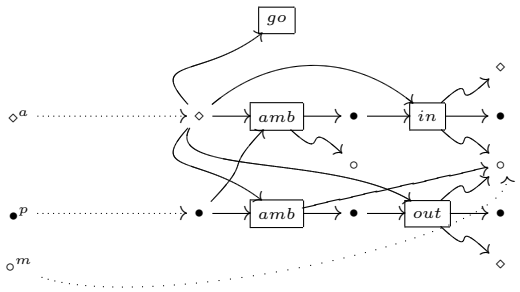


Fig. 9. Graph encoding for the process $(\nu n)(n[in\ m.\mathtt{0}]|m[out\ m.\mathtt{0}])$.

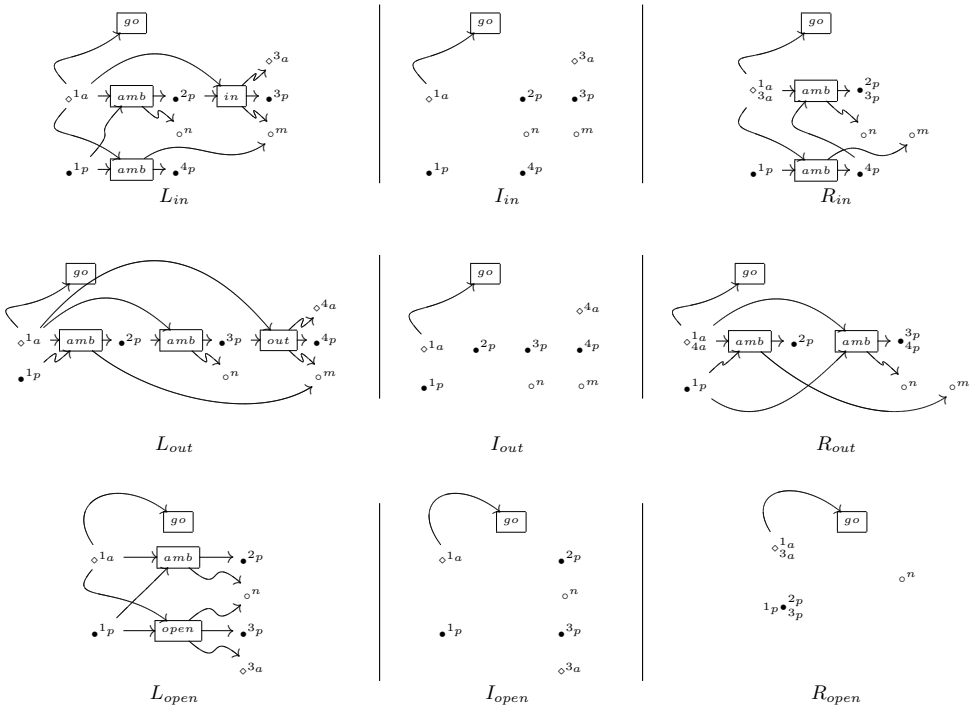


Fig. 10. The rewriting rules p_{in}, p_{out}, and p_{open}.

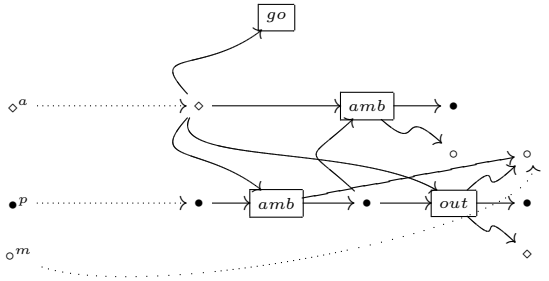


Fig. 11. Graph encoding for the process $(\nu n)(m[n[\mathsf{0}]|out\ m.\mathsf{0}])$.

Example 6.1. Let us consider again the process $P = (\nu n)(n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}])$ from Example 2.1: its enriched graphical encoding is in Fig. 9. The edge labelled *go* denotes the entry point for the computation of the process. All edges of the graph can be involved in a reduction step because they have the same activation node with an outgoing *go* edge.

The application of the p_{in} rule to the graph in Fig. 9 results in the graph in Fig. 11, the encoding of process $(\nu n)(m[n[\mathsf{0}]|out\ m.\mathsf{0}])$. This rewriting step simulates the transition $P \to (\nu n)(m[n[\mathsf{0}]|out\ m.\mathsf{0}])$ in Example 2.1. With respect to $Q = n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}] \to m[n[\mathsf{0}]|out\ m.\mathsf{0}]$, the application of the (ResRed) rule is immaterial: the occurrence of the restriction operator is mimicked by embedding L_{in} into a graph lacking n in the interface.

7. The Synthesized Transition System

This section applies the BC synthesis mechanism to $\mathcal{R}_{amb}$ in order to derive a LTS for graphs representing MAs processes. We open with an introductory section explaining the graphical counterpart of process variables (Section 7.1): these are used in the presentation (Section 7.2) of some rewriting steps with BCs. We then introduce (Section 7.4) a compact representation of the derived LTS by means of *minimal derivations*: these are extrapolated via pruning techniques (Section 7.3). The resulting LTS is going to be exploited in Section 8, in order to define a novel LTS directly for MAs processes.

7.1. *Process variables, graphically*

We first illustrate how a single BC transition may induce a reduction involving extended processes. To this end, consider the graph $J \rightarrowtail G$ in Fig. 12 and the diagram in Definition 4.3. The former represents the encoding of the process $S = (\nu n)(m[\mathsf{0}] \mid n[\mathsf{0}])$.

The occurrence of nodes $\bullet^{1_p}$ and $\diamond^{1_a}$ ensures that the process represented by $J \rightarrowtail F$, $T = \mathit{open}\, m.\mathsf{0}$, is put in parallel with S, so that $J \rightarrowtail G^+$ intuitively corresponds to $S \mid T$. Note however the occurrence of the nodes $\bullet^{2_p}$ and $\diamond^{2_a}$ in K: they witness the possibility of a parametric instance of process T. Indeed, the graph with interfaces $K \rightarrowtail G^+$ represents $S \mid T_X$, for any process variable X and linear process $T_X = \mathit{open}\, m.X$.

Put differently, the context $J \rightarrowtail F \leftarrowtail K$ is the minimal context allowing the reaction, because it could be further instantiated with any substitution of the process variable X.

7.2. *Examples of borrowed transitions*

This section shows the application of the BC synthesis mechanism to the graphical encoding of a process. Let us consider the graph $J \rightarrowtail G = [\![P]\!]^{go}_{\{m\}}$, where $P = (\nu n)(n[\mathit{in}\ m.\mathsf{0}] | m[\mathit{out}\ m.\mathsf{0}])$. In the following we discuss the possible transitions with source $J \rightarrowtail G$ that are induced by the rule $p_{in} : L_{in} \leftarrowtail I_{in} \rightarrow R_{in}$ of $\mathcal{R}_{amb}$. Since for each pair of monos $G \leftarrowtail D \rightarrowtail L_{in}$ a labelled transition might exist, in order to perform a complete analysis, we should consider all the pairs of monos $G \leftarrowtail D \rightarrowtail L_{in}$. We proceed by showing some of the possible transitions generated by such pairs. Actually, we are going to see that it is not necessary to check all those pairs that we are not considering here, by exploiting the pruning techniques presented in the next subsection.

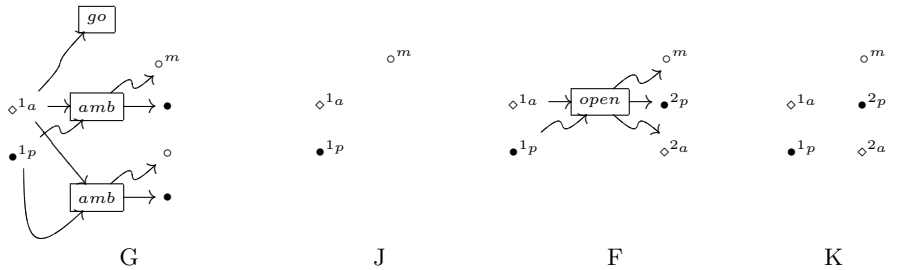


Fig. 12. The graphs with interfaces $J \rightarrowtail G$ and the context $J \rightarrowtail F \leftarrowtail K$.

BC transition for D equal to L_{in} Let D be L_{in}: there is only one map into G. The resulting transition is in Fig. 20. G^+ coincides with G, and C and H are constructed as in a standard DPO rewriting step. Since $J \rightarrowtail G$ needs no context for the reaction, the label of this transition is the identity context: two isomorphisms into the discrete graphs with three nodes $\{p, a, m\}$ (i.e., the value of the expression $id_p \otimes id_a \otimes id_m$, see Section 5). Intuitively, this corresponds to an internal transition over processes, labelled with τ.

BC transition for D equal to the upper sub-graph of L_{in} Let D be the sub-graph of L_{in} representing an ambient with a capability *in* inside it: again, there is only one map into G. The resulting transition is in Fig. 21. G^+ coincides with G in parallel with the graph representing an ambient m, encoding the process $(\nu n)(n[in\ m.0]|m[out\ m.0]|m[X])$ for some variable X. In order to reach G^+, $J \rightarrowtail G$ borrows from the environment the context $J \rightarrowtail F \leftarrowtail K$ (the syntactic context $-|m[X]$). In the interface K there is a node $\bullet^{4_p}$ pointing to the process node of F inside the ambient m, and this node represents a variable X, as detailed in Section 7.1. C and H are then constructed as in the standard DPO approach. Intuitively, $K \rightarrow H$ represents the process $m[out\ m.0]|m[n[0]|X]$, where X is the variable occurring in the label $J \rightarrowtail F \leftarrowtail K$. This can be understood by observing that the node $\bullet^{4_p}$ of K points both to a node of H and to a node of F. Summarizing, the ambient n moves into an ambient m that is provided by the environment.

BC transition for D equal to the lower sub-graph of L_{in} Let D be the lower sub-graph of L_{in} consisting of the ambient edge alone. In this case, there are two possible maps into G: the map into the sub-graph of G representing the ambient m, and the map into the sub-graph of G representing the restricted ambient n.

In the first case, we obtain the transition in Fig. 22. G^+ coincides with G in parallel with the graph representing a fresh ambient name w having inside a capability *in* m, encoding the process $(\nu n)(n[in\ m.0]|m[out\ m.0]|w[in\ m.X_2|X_1])$ for some variables X_1, X_2. In order to reach G^+, $J \rightarrowtail G$ borrows from the environment the context $J \rightarrowtail F \leftarrowtail K$ (the syntactic context $-|w[in\ m.X_2|X_1]$). As before, X_1 and X_2 are variables, since in the interface K there are nodes $\bullet^{2_p}$ and $\bullet^{3_p}$. C and H are obtained by a standard DPO derivation. Intuitively, $K \rightarrow H$ represents the process $(\nu n)(n[in\ m.0]|m[out\ m.0|w[X_2|X_1]])$. Summarizing, an ambient w from the environment enters the ambient m of a process P.

In the second case no transition is possible. G^+ coincides with G in parallel with a fresh ambient w having inside a capability *in* n, yet the pushout complement of $J \rightarrowtail G \rightarrowtail G^+$ does not exist: n is restricted and thus it does not belong to the interface J. Intuitively, no ambient from the environment can enter inside a restricted sibling ambient n.

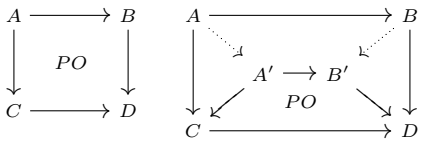
7.3. *Reducing the borrowing*

In order to derive the transitions originating from a graph $J \rightarrowtail G$, all the sub-graphs D's of L_{in}, L_{out} and L_{open} should be analyzed. To shorten this procedure, we use two pruning techniques presented in (BGK06). One exploits the observation that those items of L that are not in D must be glued to G through J. Consider a node n of D mapped to a node n' in L, such that n' is the source or the target of an edge e not in D.

Such graph items, called *boundary nodes*, are modelled via *initial pushouts* (EEPT06).

Definition 7.1 (Initial pushout). Let the square (1) below be a pushout. It is an *initial pushout* of $C \to D$ if for every other pushout as in diagram (2) there exist two unique morphisms $A \to A'$ and $B \to B'$ such that diagram (2) commutes.

Since our chosen category of typed (hyper-)graphs has initial pushouts for all arrows (EEPT06), the previous discussion can be formalized by the lemma stated below (BGK06, Corollary 1).



Lemma 7.1. A graph with interfaces $J \to G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exist a mono $D \rightarrowtail L$ (where L is the left hand side of some production in $\mathcal{R}_{amb}$), a mono $D \rightarrowtail G$, and a morphism $J_D \to J$ such that square (1) in Fig. 13 is an initial pushout and square (2) commutes.

This lemma allows to heavily prune the space of possible D's. For instance, we can exclude those D's having a continuation node (any process node depicted by $\bullet$ that is not the root) as boundary node, observing that the only process node in the interface J is the root node. We can also exclude those D's having as boundary node a name node that is not in the interface J, otherwise J_D should contain such a node and the morphism $J_D \to J$ would not exist. Consider e.g. the p_{open} rule: this means that when the node n is not in the interface J (the name n of the encoded process is bound), then $J \to G$ can perform a BC rewriting step if the node n in D is not boundary, i.e., if D contains both the *open* and the *amb* edges.

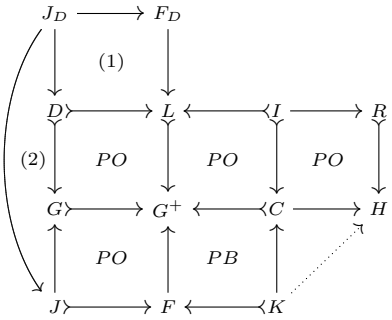


Fig. 13. BC construction with commuting squares (1) (the initial pushout of $D \rightarrowtail L$) and (2).

A further pruning —partially based on proof techniques presented in (EK06)— is performed by excluding those D's which generate a BC transition that is not relevant for bisimilarity. We may e.g. exclude those D's that contain only nodes, since they can be embedded in every graph (with the same interface) generating the same transitions. Concerning our case study, also those transitions generated by a D having the root node without the *go* edge are not relevant: a graph can perform a BC transition using such a D if and only if it can perform a transition using the same D with a *go* edge outgoing from the root. Note indeed that the resulting states of these two transitions only differ for the number of *go* edges attached to the root: the state resulting after the first transition has two *go*'s, the state resulting after the second transition only one. These states are bisimilar, since the number of *go*'s does not change the behavior (BGK06, Lemma 12).

The two pruning techniques presented above allow us to only consider the partial matches D shown in Figs. 15, 23 and 24, together with those D's obtained from the ones of the last two figures by coalescing the name nodes n and m.

7.4. *Minimal transitions*

In Section 7.3 we restricted a lot the space of possible D's. However, reasoning on the synthesized LTS is still hard (this is usually with derived LTSs, as pointed out in (BEK06) and (Bon08), where an SOS presentation of the synthesized LTS is deemed as desirable). In order to simplify this reasoning, we introduce a set of *minimal transitions* that allow us to derive all and only the transitions of the (pruned) synthesized LTS.

Inspired by Lemma 7.1, providing necessary and sufficient conditions for performing a transition, we consider the graphs $J_D \to D$ for all those D's that have not been pruned in Section 7.3 and J_D containing only the boundary nodes of D.

The minimal transitions have the shape as in Fig. 14, where the leftmost square in the lower row is an initial pushout. Figs. 15, 23 and 24 concisely represent these transitions, showing their starting graph D, the label $J_D \rightarrowtail F_D \leftarrowtail K_D$, and the resulting graph R. The three figures represent the minimal transitions generated by the rules p_{open}, p_{in} and p_{out}. The minimal transitions generated by the rules p_{in-c} and p_{out-c} should also be considered, but they are easily described from those of p_{in} and p_{out}. More precisely, for each minimal transition with D_{in_x} there exists a minimal transition generated by p_{in-c}, where the relevant graphs are obtained by coalescing the nodes n and m (and similarly for D_{out_x} and p_{out-c}).[§]

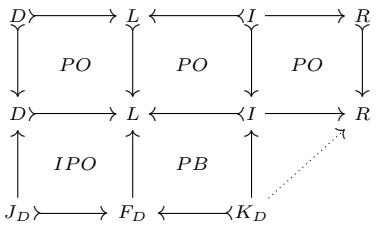


Fig. 14. Transition shape.

All the transitions originating from a process encoding $J \rightarrowtail G$ can be characterized via these minimal transitions. By Lemma 7.1, we can state that $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exist a mono $D \rightarrowtail G$, for some D of the minimal transitions, and a morphism $J_D \to J$ such that square (2) in Fig. 13 commutes.

The label of the rewriting step can be obtained from the label of the minimal transition. First of all note that the interface J contains all the nodes of J_D (as suggested by the morphism $J_D \to J$), all the name nodes $\circ$ representing the free names of the modeled process (as expected by our encoding), and the root nodes of the graph D when they are not in J_D. Then the graph F only contains the whole graph F_D and all the nodes of J. Indeed, as shown in the proposition below, which is an adaptation of Proposition 4 of (BGK06), F can be obtained as the pushout of $J_D \to F_D$ and $J_D \to J$.

Proposition 7.1. Let $p : L \leftarrowtail I \to R$ be a production of $\mathcal{R}_{amb}$, $J \rightarrowtail G$ a graph with interfaces, and $d : D \rightarrowtail L$ a mono such that square (i) in Fig. 16 is the initial pushout of d and square (ii) is a pullback. There is a K such that $J \rightarrowtail G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$ *via* p and d if and only if there are a mono $D \rightarrowtail G$, a graph V and a morphism $J_D \to J$ such that diagram (iii) in Fig. 16 commutes and F and H are built as shown there.

It is easy to prove that K is a discrete graph containing exactly the nodes of F (i.e, K consists only of the nodes of J and K_D).

[§] Note also that it is irrelevant to consider the coalesced version for the rule with D'_{in_i}, since it would coincide with the minimal transition for D_{in-c_i}, for all i.

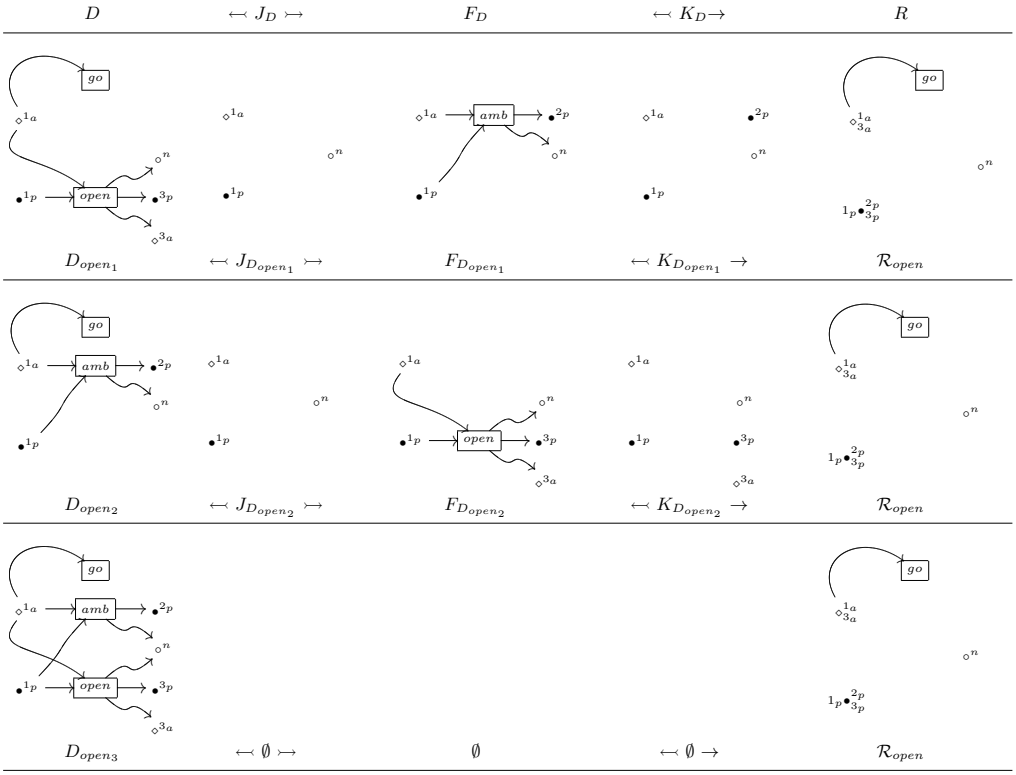


Fig. 15. The minimal transitions generated by the rule p_{open}.

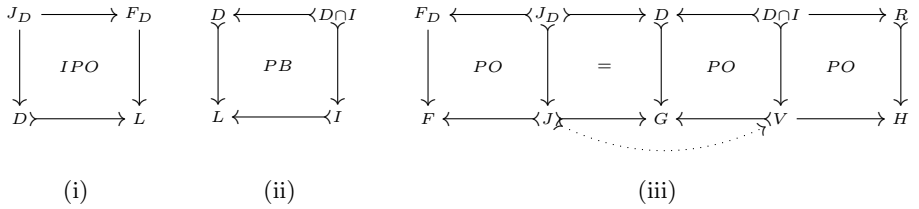


Fig. 16. Diagrams used in Proposition 7.1.

Finally, the resulting graph H is obtained by replacing in G the sub-graph D with R. As shown in Proposition 7.1, it can be computed in a DPO step of $D \leftarrowtail D \cap I \rightarrow R$, where $D \cap I$ is the pullback of $D \rightarrowtail L$ and $I \rightarrowtail L$.

As an example, consider the BC rewriting step in Fig. 20. It is derivable by the minimal transition for D_{in_4} (in Fig. 23). First of all, note that there exist $D_{in_4} \rightarrow G$ and $\emptyset \rightarrow J$ such that the square (2) in Fig. 13 commutes. Now, F is equal to J, since it consists of the composition of $F_{D_{in_4}}$ (i.e., $\emptyset$) and J. The new interface K is equal to F, since it contains all and only the nodes of J and $K_{D_{in_4}}$ (i.e., $\emptyset$). The arriving state H is obtained simply by replacing D_{in_4} with R_{in}.

$$\text{(InTau)}\quad \frac{P\equiv(\nu A)\ \mathcal{C}[n[in\ m.P_1|P_2]|m[P_3]]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[m[n[P_1|P_2]|P_3]]} \qquad \text{(OutTau)}\quad \frac{P\equiv(\nu A)\ \mathcal{C}[m[n[out\ m.P_1|P_2]|P_3]]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[m[P_3]|n[P_1|P_2]]}$$

$$\text{(OpenTau)}\ \frac{P\equiv(\nu A)\ \mathcal{C}[n[P_1]|open\ n.P_2]}{P\xrightarrow{-}(\nu A)\ \mathcal{C}[P_1|P_2]}$$

Fig. 17. The internal transitions of the LTS $\mathcal{D}$ (for $\mathcal{C}[-]$ an enabling context).

8. A New LTS for Mobile Ambients

This section presents the LTS $\mathcal{D}$ directly defined over MAs processes. The inference rules describing this LTS are obtained from the transitions of the LTS on graphs presented in Section 7.4. The labels of the transitions are unary contexts, i.e., terms of the extended syntax with a hole $-$. The formal definition of our LTS is shown in Figs. 17 and 18.

8.1. *The labelled rules on processes...*

The rules in Fig. 17 represent the τ-actions modeling internal computations. The labels of the transitions are contexts composed of just a hole $-$, while the resulting states are pure processes. Also, an *enabling* context may contain only ambients and parallel operators The rule InTau enables an ambient n to enter a sibling ambient m. The rule OutTau enables an ambient n to get out of its parent ambient m. Finally, the rule OpenTau models the opening of an ambient n. These three rules exactly derive the MAs reduction relation, thus they could be replaced with the rules in Table 10.

The rules in Fig. 18 model the interactions of a process with its environment, and labels and resulting states contain process and name variables. We define the LTS $\mathcal{D}_{\mathcal{I}}$ for MAs pure processes by instantiating all those variables.

Definition 8.1. Let P, Q be pure processes and let $C[-]$ be a pure context. We have a transition $P \xrightarrow{C[-]}_{\mathcal{D}_{\mathcal{I}}} Q$ if there exist a transition $P \xrightarrow{C_\epsilon[-]}_{\mathcal{D}} Q_\epsilon$ and a substitution σ such that $Q_\epsilon\sigma \equiv Q$ and $C_\epsilon[-]\sigma = C[-]$.

Recall that substitutions map name variables into ambient names and process variables into pure processes, and that they may never capture bound names.

$$\text{(In)}\quad \frac{P\equiv(\nu A)(in\ m.P_1|P_2)\quad m\notin A}{P\xrightarrow{x[-|X_1]|m[X_2]}(\nu A)m[x[P_1|P_2|X_1]|X_2]} \qquad \text{(OutAmb)}\quad \frac{P\equiv(\nu A)(n[out\ m.P_1|P_2]|P_3)\quad m\notin A}{P\xrightarrow{m[-|X_1]}(\nu A)(m[P_3|X_1]|n[P_1|P_2])}$$

$$\text{(InAmb)}\ \frac{P\equiv(\nu A)(n[in\ m.P_1|P_2]|P_3)\quad m\notin A}{P\xrightarrow{-|m[X_1]}(\nu A)(m[n[P_1|P_2]|X_1]|P_3)} \qquad \text{(Open)}\quad \frac{P\equiv(\nu A)(open\ n.P_1|P_2)\quad n\notin A}{P\xrightarrow{-|n[X_1]}(\nu A)(P_1|X_1|P_2)}$$

$$\text{(CoIn)}\quad \frac{P\equiv(\nu A)(m[P_1]|P_2)\quad m\notin A}{P\xrightarrow{-|x[in\ m.X_1|X_2]}(\nu A)(m[x[X_1|X_2]|P_1]|P_2)} \qquad \text{(CoOpen)}\quad \frac{P\equiv(\nu A)(n[P_1]|P_2)\quad n\notin A}{P\xrightarrow{-|open\ n.X_1}(\nu A)(P_1|X_1|P_2)}$$

$$\text{(Out)}\quad \frac{P\equiv(\nu A)(out\ m.P_1|P_2)\quad m\notin A}{P\xrightarrow{m[x[-|X_1]|X_2]}(\nu A)(m[X_2]|x[P_1|P_2|X_1])}$$

Fig. 18. The environmental transitions of the LTS $\mathcal{D}$.

The rule Open models the opening of an ambient provided by the environment. More explicltly, it enables a process P with a capability *open* $n.P_1$ at top level, for $n \in fn(P)$, to interact with a context providing an ambient n that contains inside it some process X_1. The resulting state is the process over the extended syntax $(\nu A)(P_1|X_1|P_2)$, where X_1 represents a process provided by the environment. Note that the instantiation of the process variable X_1 with a process containing a free name that belongs to the bound names in A is possible only α-converting the resulting process $(\nu A)(P_1|X_1|P_2)$ into a process that does not contain that name among its bound names at top level.

The rule CoOpen instead models an environment that opens an ambient of the process. The rule InAmb enables an ambient of the process to migrate into a sibling ambient provided by the environment, while in the rule In both ambients are so. In the rule CoIn an ambient provided by the environment enters an ambient of the process. In the rule OutAmb an ambient of the process exits from an ambient provided by the environment, while in the rule Out both ambients are so.

The LTS $\mathcal{D}$ does not conform to the SOS style, since the premises of the inference rules are just constraints over the process structure. This is due to the fact that the rules of the LTS $\mathcal{D}$ are derived from the minimal transitions. Each rule corresponds to one minimal transition presented in Section 7.4 and it is obtained as described below.

8.2. *...from the borrowed rules on graphs*

Observe that a graph $J \rightarrowtail G$ representing a process P can perform a BC rewriting step in $\mathcal{R}_{amb}$ if and only if there exists a mono $D \rightarrowtail G$, for some D of a minimal transition, and a morphism $J_D \to J$, such that the square (2) in Fig. 13 commutes. Moreover, the label and the resulting graph of the borrowed transition for G are obtained from the label and the resulting state of the minimal transition of D, respectively. Therefore, for each minimal transition we obtain an inference rule: the conditions in the premise correspond to the necessary and sufficient conditions for performing a transition from a graph G, while the label and the resulting process are obtained from the label and the resulting state of the borrowed transition, respectively. Since the labels of the LTS over graphs obtained by the BC mechanism represent minimal graph contexts enabling a graph production, the labels of our LTS over processes represent minimal process contexts enabling a reduction.

As the main example, in this section we closely look at the correspondence between the rule Open and the first minimal transition in Fig. 15.

Consider a graph $J \rightarrowtail G$ representing the encoding for a process P. If there exists a mono $D_{open_1} \rightarrowtail G$ and a morphism $J_{D_{open_1}} \to J$, such that the square (2) in Fig. 13 commutes, the graph $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ with label $J \rightarrowtail F \leftarrowtail K$, where J, F and K respectively consist of $J_{D_{open_1}}$, $F_{D_{open_1}}$ and $K_{D_{open_1}}$ together with the free names of P. Now, note that D_{open_1} can be embedded in G and a morphism $J_{D_{open_1}} \to J$ (such that the square (2) in Fig. 13 commutes) may exist if and only if $P \equiv (\nu A)(open\ n.P_1|P_2)$, for $n \notin A$. Indeed, the graph must contain an occurrence of the operator *open* $n.-$ on top, possibly further instantiated, since it includes D_{open_1}; and since the interface J contains all the nodes of $J_{D_{open_1}}$, we conclude that n must belong to J, that is, n must be a free name of P. This is the premise of the rule Open.

Starting from the label $J \rightarrowtail F \leftarrowtail K$ of the BC transition we now obtain the label of the process transition. By observing the shape of F, which contains all the items of $F_{D_{open_1}}$, we can say that the process context is composed of the ambient n. Moreover, the context F is glued to G through J, which contains the free names of P and the nodes of $J_{D_{open_1}}$, i.e., the name n and the nodes representing the roots of the graph G (which models P). Since these two nodes represent the roots of the graph F (which models ambient n), we conclude that the label of the process transition is a context with the ambient n in parallel with a hole representing process P.

The graph K represents the interface of both graphs F and H. It contains all the nodes of $K_{D_{open_1}}$, i.e., the roots of F and the roots of the process inside the ambient n. The nodes of the interface K represent the "handles" of F and H for interacting with an environment. Therefore, the process node of K that is not the root of F can be thought of as a process variable inside the ambient n in the label of the transition. Therefore, we conclude that the label of the transition with source the process P can be represented as the minimal context $-|n[X_1]$, where $-$ is a hole and X_1 is a process variable. The resulting process $(\nu A)(P_1|X_1|P_2)$ exactly corresponds to the state H from the BC transition. Indeed, in the interface K of the graph $K \rightarrow H$ also the node modeling the process variable X_1 occurs, which represents a process provided by the environment.

The reader should notice that while there are 13 minimal transitions, only 10 rules occur in Figs. 17 and 18. This is due to the fact that each of the rules In, CoIn and Out is actually derived by two minimal transitions. The rule In is generated by the minimal transitions D_{in_1} and D'_{in_1}, CoIn by D_{in_3} and D'_{in_3}, and Out by D_{out_1} and D'_{out_1}. We show the latter, since the others are analogous.

In the minimal transition with D_{out_1} two ambients are borrowed from the environment. The first one has name m (i.e., the ambient from which the process wants to exit), while the second has a fresh name n (it is not restricted, since it occurs in $K_{D_{out_1}}$). This transition thus corresponds to the rule

$$\frac{P \equiv (\nu A)(out\ m.P_1|P_2) \quad m \notin A \quad n \notin (A \cup \boldsymbol{fn}(P))}{P \xrightarrow{m[n[-|X_1]|X_2]} (\nu A)(m[X_2]|n[P_1|P_2|X_1])}$$

In the minimal transition with D'_{out_1} the name n belongs to the process (it occurs inside the graph $D_{out'_1}$) but, since the node n occur in $J_{D'_{out_1}}$, it should appear in the interface J, i.e., it must be free. Thus, this transition corresponds to the rule

$$\frac{P \equiv (\nu A)(out\ m.P_1|P_2) \quad m \notin A \quad n \in \boldsymbol{fn}(P)}{P \xrightarrow{m[n[-|X_1]|X_2]} (\nu A)(m[X_2]|n[P_1|P_2|X_1])}$$

The conclusion of the two rules above is identical, thus we can put together their premises, and compactly represent them via the rule Out of Fig. 18. Substituting the name n with a name variable x basically guarantees that any actual name can be substituted to n, even m (thanks to D_{out-c_1}), as long as it does not occur in A.

9. A SOS Presentation for the Derived LTS $\mathcal{D}$

In the previous two sections we described a semi-automatic methodology for distilling a LTS $\mathcal{D}$. This section introduces a set of SOS rules, tailored over $\mathcal{D}$, such that the associated LTS $\mathcal{S}$ coincides with the former one. The rules for $\mathcal{S}$ are shown in Fig. 19. We assume the implicit presence of the rule $(P \equiv P', P' \xrightarrow{C_\epsilon[-]} Q_\epsilon) \Rightarrow P' \xrightarrow{C_\epsilon[-]} Q_\epsilon$.

The rules in the first two rows of Fig. 19 model internal computations: they are obtained from the rules in Fig. 17. Since these rules derive the same transition relation of the reduction relation over MAs, we replace it with the reduction rules labelled with the identity context $-$. So, we obtain the axioms modelling the execution of the capabilities of the calculus, and a structural rule for each ambient, parallel and restriction operators.

The remaining rules in Fig. 19, modelling the interactions of a process with its environment, are obtained from the rules in Fig. 18. More explicitly, for each one of these latter we derive three rules. First, we determine the axiom by considering the minimal process needed by the reduction to occur. For exemple, with respect to the rule In of the LTS $\mathcal{D}$, the minimal process allowing the reduction is $in\ m.P_1$, and we thus determine the axiom $in\ m.P_1 \xrightarrow{x[-|X_1]|m[X_2]} m[x[P_1|X_1]|X_2]$. The next step consists in determining the relative structural rules in SOS style. Once more, as far as the rule In of the LTS $\mathcal{D}$ is concerned, if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$, then for the process $P|Q$ there is a transition labelled $x[-|X_1]|m[X_2]$ leading to the process P_ϵ with the process Q inside the ambient x, that is, we have the transition $P|Q \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon\{^{Q|X_1}/_{X_1}\}$. Instead, if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$ and $m \neq a$, then we have the transition $(\nu a)P \xrightarrow{x[-|X_1]|m[X_2]} (\nu a)P_\epsilon$.

This result is also confirmed by the analysis of the minimal transitions.

Deriving axioms As explained in Section 7.4, a minimal transition represents a BC transition, where the starting graph is the smallest graph allowing a BC rewriting with respct to a given rule and a given partial match. The graph D of a minimal transition thus represents the minimal process needed to the reduction modeled by the BC transition to occur. This means that each minimal transition represents an axiom of the SOS LTS.

Let us consider for example the minimal transition for D_{in_1}. The graphs D_{in_1} represents the process $in\ m.\mathtt{0}$, but all the remarks made below also hold for the extended process $in\ m.P_1$, where P_1 represents any process. As explained in Section 7.4, starting from the label of the BC transition we obtain the label of the process transition that in this case is $x[-|X_1]|m[X_2]$, with x name variable. The resulting process is instead represented by the graph R_{in} that models the process $x[P_1|X_1]|m[X_2]$. Therefore, this minimal transition represents the axiom $in\ m.P_1 \xrightarrow{x[-|X_1]|m[X_2]} m[x[P_1|X_1]|X_2]$.

Now, let us consider the similar minimal transition for D_{in_2}. It represents the axiom $n[(\nu A)(in\ m.P_1|P_2)] \xrightarrow{-|m[X_2]} m[n[(\nu A)(P_1|P_2)]|X_2]$. Nevertheless, it is obvious that this rule can be rewritten as the rule InAmb of Fig. 19, by using the transition derived according to the rules In, InPar and InRes. Graphically, this is suggested by the fact that the graph D_{in_2} contains the partial match D_{in_1}, which gives rise to the minimal transition allowing us to derive the rule of the third row of Fig. 19.

Deriving rules for the parallel operator The structural rules can be obtained by analyzing the interface J_D of the minimal transition, whose nodes represent the "handles" of D for interacting with the environment. Since in a minimal transition J_D contains the root nodes of D, we can add a graph I representing a process Q in parallel with D, obtaining a graph $J \rightarrowtail G$ where J consists of J_D plus the free names of Q. Now, since there are a mono $D \rightarrowtail G$ (since G consists of the graph D in parallel with I) and a morphism $J_D \to J$ such that the square (2) in Fig. 13 commutes, the graph $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ with label $J \rightarrowtail F \leftarrowtail K$, where F and K consist of F_D and K_D plus the free names of Q. Process-wise, this means that if $P \xrightarrow{C[-]} P\epsilon$, then also a transition labelled $C[-]$ originating from $P|Q$.

Let us consider again the minimal transition for D_{in_1}. By analyzing the interface $J_{D_{in_1}}$ we may obtain the structural rule for the parallel operator. Since $J_{D_{in_1}}$ contains the root nodes of D, we can add a graph I representing a process Q in parallel with D, obtaining a graph $J \rightarrowtail G$ where J consists of $J_{D_{in_1}}$ plus the free names of Q. The graph $J \rightarrowtail G$ can perform a BC rewriting step in $\mathcal{R}_{amb}$ with label $J \rightarrowtail F \leftarrowtail K$, where F and K consist of $F_{D_{in_1}}$ and $K_{D_{in_1}}$ plus the free names of Q. This means that the graph context $J \rightarrowtail F \leftarrowtail K$ also represents the process context $x[-|X_1]|m[X_2]$. Process-wise, this means that if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$, then also a transition labeled with $x[-|X_1]|m[X_2]$ originates from $P|Q$. The resulting process is represented by the graph H, obtained by replacing D_{in_1} with R_{in} in G. Note that the node $\bullet_{2_p}$ after the reduction is under the ambient x and moreover it represents a process variable in P_ϵ. This means that the graph modeling Q (that has as root the node $\bullet_{2_p}$) after the reduction is under the ambient x and it is in parallel with the process variable X_1. Therefore the resulting process is $P_\epsilon\{^{Q|X_1}/_{X_1}\}$.

Deriving rules for the restriction operator Also the structural rules for the restriction operator can be obtained by analyzing the interface J_D. Indeed, we know that a graph $J \rightarrowtail G$ representing a process P can perform a BC rewriting step if and only if there exist a mono $D \rightarrowtail G$ and a morphism $J_D \to J$ such that the square (2) in Fig. 13 commutes. If we modify the interface J by removing one or more name nodes, then the graph G with the new interface J' can also perform the same BC rewriting step if and only if there exists a morphism $J_D \to J'$ such that square (2) in Fig. 13 commutes. This means that all the name nodes of J_D must also belong to J', therefore as suggested by the encoding, the ambient names of P that are in J_D cannot be restricted. In terms of processes, this means that if $P \xrightarrow{C[-]} P\epsilon$, then for the process $(\nu a)P$ there is also a transition labelled $C[-]$ if the names belonging to J_D do not belong to a.

On the basis of the remarks above, starting from the minimal transition for D_{in_1}, we can derive a structural rule for the restriction operator. In particular, if $P \xrightarrow{x[-|X_1]|m[X_2]} P\epsilon$, then for the process $(\nu a)P$ there is a transition with the same label leading to the process $(\nu a)P_\epsilon$, if the name m (that belongs to the interface $J_{D_{in_1}}$) is not restricted.

Note that the interface J_D also allows us to obtain a graph $J \rightarrowtail G$ that is composed of the graph D with another graph on the top. However, it is easy to note that in this case the graph $J \rightarrowtail G$ does not perform any BC transition because it is impossible to find a morphism $J_D \to J$ such that square (2) in Fig. 13 commutes.

(InTau)
$$\frac{}{n[in\ m.P|Q]|m[R] \xrightarrow{-} m[n[P|Q]|R]}$$

(OutTau)
$$\frac{}{m[n[out\ m.P|Q]|R] \xrightarrow{-} n[P|Q]|m[R]}$$

(OpenTau)
$$\frac{}{open\ n.P|n[Q] \xrightarrow{-} P|Q}$$

(TauAmb)
$$\frac{P \xrightarrow{-} P'}{n[P] \xrightarrow{-} n[P']}$$

(TauPar)
$$\frac{P \xrightarrow{-} P'}{P|Q \xrightarrow{-} P'|Q}$$

(TauRes)
$$\frac{P \xrightarrow{-} P'}{(\nu a)P \xrightarrow{-} (\nu a)P'}$$

(In)
$$\frac{}{in\ m.P_1 \xrightarrow{x[-|X_1]|m[X_2]} m[x[P_1|X_1]|X_2]}$$

(InPar)
$$\frac{P \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon}{P|Q \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon\{^{Q|X_1}/_{X_1}\}}$$

(InRes)
$$\frac{P \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon \quad a \neq m}{(\nu a)P \xrightarrow{x[-|X_1]|m[X_2]} (\nu a)P_\epsilon}$$

(InAmb)
$$\frac{P \xrightarrow{x[-|X_1]|m[X_2]} P_\epsilon}{n[P] \xrightarrow{-|m[X_2]} P_\epsilon\{^{n}/_{x},^{0}/_{X_1}\}}$$

(InAmbPar)
$$\frac{P \xrightarrow{-|m[X_2]} P_\epsilon}{P|Q \xrightarrow{-|m[X_2]} P_\epsilon|Q}$$

(InAmbRes)
$$\frac{P \xrightarrow{-|m[X_2]} P_\epsilon \quad a \neq m}{(\nu a)P \xrightarrow{-|m[X_2]} (\nu a)P_\epsilon}$$

(CoIn)
$$\frac{}{m[P_1] \xrightarrow{-|x[in\ m.X_1|X_2]} m[x[X_1|X_2]|P_1]}$$

(CoInPar)
$$\frac{P \xrightarrow{-|x[in\ m.X_1|X_2]} P_\epsilon}{P|Q \xrightarrow{-|x[in\ m.X_1|X_2]} P_\epsilon|Q}$$

(CoInRes)
$$\frac{P \xrightarrow{-|x[in\ m.X_1|X_2]} P_\epsilon \quad a \neq m}{(\nu a)P \xrightarrow{-|x[in\ m.X_1|X_2]} (\nu a)P_\epsilon}$$

(Out)
$$\frac{}{out\ m.P_1 \xrightarrow{m[x[-|X_1]|X_2]} m[X_2]|x[P_1|X_1]}$$

(OutPar)
$$\frac{P \xrightarrow{m[x[-|X_1]|X_2]} P_\epsilon}{P|Q \xrightarrow{m[x[-|X_1]|X_2]} P_\epsilon\{^{Q|X_1}/_{X_1}\}}$$

(OutRes)
$$\frac{P \xrightarrow{m[x[-|X_1]|X_2]} P_\epsilon \quad a \neq m}{(\nu a)P \xrightarrow{m[x[-|X_1]|X_2]} (\nu a)P_\epsilon}$$

(OutAmb)
$$\frac{P \xrightarrow{m[x[-|X_1]|X_2]} P_\epsilon}{n[P] \xrightarrow{m[-|X_2]} P_\epsilon\{^{n}/_{x},^{0}/_{X_1}\}}$$

(OutAmbPar)
$$\frac{P \xrightarrow{m[-|X_2]} P_\epsilon}{P|Q \xrightarrow{m[-|X_2]} P_\epsilon\{^{Q|X_2}/_{X_2}\}}$$

(OUtAmbRes)
$$\frac{P \xrightarrow{m[-|X_2]} P_\epsilon \quad a \neq m}{(\nu a)P \xrightarrow{m[-|X_2]} (\nu a)P_\epsilon}$$

(Open)
$$\frac{}{open\ n.P_1 \xrightarrow{-|n[X_1]} P_1|X_1}$$

(OpenPar)
$$\frac{P \xrightarrow{-|n[X_1]} P_\epsilon}{P|Q \xrightarrow{-|n[X_1]} P_\epsilon|Q}$$

(OpenRes)
$$\frac{P \xrightarrow{-|n[X_1]} P_\epsilon \quad a \neq n}{(\nu a)P \xrightarrow{-|n[X_1]} (\nu a)P_\epsilon}$$

(CoOpen)
$$\frac{}{n[P_1] \xrightarrow{-|open\ n.X_1} P_1|X_1}$$

(CoOpenPar)
$$\frac{P \xrightarrow{-|open\ n.X_1} P_\epsilon}{P|Q \xrightarrow{-|open\ n.X_1} P_\epsilon|Q}$$

(CoOpenRes)
$$\frac{P \xrightarrow{-|open\ n.X_1} P_\epsilon \quad a \neq n}{(\nu a)P \xrightarrow{-|open\ n.X_1} (\nu a)P_\epsilon}$$

Fig. 19. The LTS $\mathcal{S}$.

Equivalence between the LTSs As for $\mathcal{D}$, also for $\mathcal{S}$ we define the LTS $\mathcal{S}_\mathcal{I}$ for pure processes by instantiating all the variables of the labels and of the resulting states.

Definition 9.1. Let P, Q be pure processes and let $C[-]$ be a pure context. Then, we have a transition $P \xrightarrow{C[-]}_{\mathcal{S}_\mathcal{I}} Q$ if there exist a transition $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ and a substitution σ such that $Q_\epsilon\sigma \equiv Q$ and $C_\epsilon[-]\sigma = C[-]$.

As stated below, the LTSs $\mathcal{S}_I$ and $\mathcal{D}_I$ coincide (the proof is shown in Appendix A).

Theorem 9.1. Let P be a pure process and let $C[-]$ be a pure context. Then, $P \xrightarrow{C[-]}_{\mathcal{D}_\mathcal{I}} Q$ if and only if $P \xrightarrow{C[-]}_{\mathcal{S}_\mathcal{I}} Q$.

10. Equivalence between LTSs

This section shows the equivalence between our LTS $\mathcal{S}_\mathcal{I}$ defined on pure processes and the LTS proposed by Rathke and Sobociński in (RS08b, Figs. 6, 7 and 8).

Their LTS, closed under structural congruence, is split in three components: the process-view LTS $\mathcal{C}$, the context-view LTS $\mathcal{A}$, and the combined LTS $\mathcal{CA}$. The labels of the LTS $\mathcal{CA}$ have the shape $\alpha \downarrow \vec{M}$: α is derived by LTS $\mathcal{C}$ and $\vec{M}$ by LTS $\mathcal{A}$. If $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$, α identifies the minimal context needed by the pure process P to react and $\vec{M}$ is a list of pure processes and ambient names, instantiating the context components.

The first column of Table 5 shows the labels α of the LTS $\mathcal{C}$, while the second column presents the context χ_α that each label identifies (the correspondence is formally explained in (RS08b, Lemma 6)). Note that each context χ_α contains a set of typed numbered holes: those of type N can be instantiated with ambient names, those of type Pr with pure processes. Thus, in a transition $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} P'$, the tuple $\vec{M}$ instantiates all context components, that is, each hole of χ_α of process type Pr (different from 1_{Pr}) and of name type N. The hole 1_{Pr} represents the hole that has to be instantiated with the process P, and it corresponds to the hole denoted $-$ in our contexts. For example, if $P \xrightarrow{in\, m\downarrow\vec{M}}_{\mathcal{CA}} P'$, then the tuple $\vec{M}$ instantiate the holes 2_{Pr}, 3_N and 4_{Pr} of the context $\chi_{in\, m}$, i.e., it has the shape $\vec{M} : Q, n, R$, for Q, R pure processes and n ambient name.

It is immediate to note that there exists a one-to-one correspondence between the labels $C_\epsilon[-]$ of our LTS $\mathcal{S}$ and the contexts χ_α listed in the second column of Table 5. This correspondence is shown in the same table, where $C^\alpha_\epsilon[-]$ (the third column) denotes the label of our LTS $\mathcal{S}$ corresponding to the context χ_α of the second column.

For each label α, the contexts $C^\alpha_\epsilon[-]$ and χ_α have the same shape. The hole $-$ in $C^\alpha_\epsilon[-]$ corresponds to the hole 1_{Pr} in χ_α, and there is a correspondence between name and process variables of $C^\alpha_\epsilon[-]$ and holes of χ_α of type N an Pr. Consider e.g. the label $C^{in\, m}_\epsilon[-] = x[-|X_1]|m[X_2]$ and the context $\chi_{in\, m} = 3_N[1_{Pr}|2_{Pr}]|m[4_{Pr}]$: they have the same shape, and the name variable x corresponds to the hole 3_N, the hole $-$ to the hole 1_{Pr} , and the process variables X_1 and X_2 to the holes 2_{Pr} and 4_{Pr}.

As explained in Section 9, a substitution σ for a context $C_\epsilon[-]$ provides an instantiation for its variables. For instance, a substitution for $C^{in\, m}_\epsilon[-] = x[-|X_1]|m[X_2]$ must have the shape $\{^Q/_{X_1}, ^n/_x, ^R/_{X_2}\}$ for Q, R pure processes and n ambient name. Since

α	χ_α	$C^\alpha_\epsilon[-]$	$\vec{M}^\alpha_\sigma$	σ^α_M
$in\ m$	$3_N[1_{Pr}\|2_{Pr}]\|m[4_{Pr}]$	$x[-\|X_1]\|m[X_2]$	PnQ	$\{^P/_{X_1}, ^n/_x, ^Q/_{X_2}\}$
$[in\ m]$	$1_{Pr}\|m[2_{Pr}]$	$-\|m[X_2]$	P	$\{^P/_{X_2}\}$
$\overline{[in\ m]}$	$4_N[in\, m.2_{Pr}\|3_{Pr}]\|1_{Pr}$	$-\|x[in\ m.X_1\|X_2]$	PQn	$\{^P/_{X_1}, ^Q/_{X_2}, ^n/_x\}$
$out\ m$	$m[3_N[1_{Pr}\|2_{Pr}]\|4_{Pr}]$	$m[x[-\|X_1]\|X_2]$	PnQ	$\{^P/_{X_1}, ^n/_x, ^Q/_{X_2}\}$
$[out\ m]$	$m[1_{Pr}\|2_{Pr}]$	$m[-\|X_2]$	P	$\{^P/_{X_2}\}$
$open\ n$	$1_{Pr}\|n[2_{Pr}]$	$-\|n[X_1]$	P	$\{^P/_{X_1}\}$
$\overline{open\ n}$	$open\, n.2_{Pr}\|1_{Pr}$	$-\|open\ n.X_1$	P	$\{^P/_{X_1}\}$
τ	1_{Pr}	$-$	$\emptyset$	$\{\}$

Table 5. The correspondence between α, χ_α, $C^\alpha_\epsilon[-]$, $\vec{M}^\alpha_\sigma$ and σ^α_M.

there is a correspondence between holes of a context χ_α of type N and Pr, and name and process variables of the context $C^\alpha_\epsilon[-]$, any tuple $\vec{M}$ for χ_α determines a unique substitution σ_M for $C^\alpha_\epsilon[-]$, instantiating each variable with the value used by $\vec{M}$ to instantiate the hole corresponding to that variable. Analogously, a substitution σ for $C^\alpha_\epsilon[-]$ determines a unique substitution $\vec{M}_\sigma$ for χ_α. Consider again the context $\chi_{in\, m}$ and the tuple $\vec{M} = Q, n, R$ providing an instantiation for the holes 2_{Pr}, 3_N and 4_{Pr}. The substitution σ_M (induced by $\vec{M}$) for the context $C^{in\, m}_\epsilon[-] = x[-|X_1]|m[X_2]$ is $\{^Q/_{X_1}, ^n/_x, ^R/_{X_2}\}$. Analogously, it is possible to determine the tuple $\vec{M}$ from the substitution σ_M. The last two columns of Table 5 show for each α the shape of the tuples $\vec{M}^\alpha_\sigma$ and σ^α_M.

Proposition 10.1. Let P be a pure process. If $P \xrightarrow{\alpha\downarrow\vec{M}^\alpha}_{\mathcal{CA}} Q$, then there exists Q_ϵ such that $P \xrightarrow{C^\alpha_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ and $Q \equiv Q_\epsilon\sigma^\alpha_M$.

Proposition 10.2. Let P be a pure process and let σ be a substitution. If $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ and $Q_\epsilon\sigma \equiv Q$, then there exists α such that $C_\epsilon[-] = C^\alpha_\epsilon[-]$ and $P \xrightarrow{\alpha\downarrow\vec{M}^\alpha_\sigma}_{\mathcal{CA}} Q$.

From the two propositions above (their proofs are in Appendix B) and from the definition of the LTS $\mathcal{S}_\mathcal{I}$ (Definition 9.1) follows the main result of this section.

Theorem 10.1. Let P be a pure process. If $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$, then there is a unique (up-to $\equiv$) substitution σ such that $P \xrightarrow{C^\alpha_\epsilon[-]\sigma}_{\mathcal{S}_\mathcal{I}} Q$. Vice versa, if $P \xrightarrow{C[-]}_{\mathcal{S}_\mathcal{I}} Q$, then there are α and a unique (up-to $\equiv$) tuple $\vec{M}$ such that $C[-] = C^\alpha[-]$ and $P \xrightarrow{\alpha\downarrow\vec{M}}_{\mathcal{CA}} Q$.

11. Conclusions, related and future work

This paper exploits a graphical encoding for MAs (GM08) to distill a LTS on (processes encoded as) graphs. This LTS is obtained semi-automatically by applying the BC technique to the GTS associated to the calculus, after using two pruning techinques for removing some reductions. The LTS defined on graphs is then used in order to infer a novel LTS $\mathcal{D}$ directly defined on MAs processes. Moreover, a set of SOS rules for MAs is presented, showing that the LTS $\mathcal{S}_I$ they induce on pure processes coincides with $\mathcal{D}_I$. Finally, we prove that our $\mathcal{S}_I$ is equivalent with an alternative proposal in (RS08b).

For the sake of simplicity, we considered the finite, communication-free fragment of MAs. A graphical encoding for the whole calculus could be obtained along the lines of the solution in (BGK06). Should the encoding for the whole calculus be defined, the technique presented in this paper could be applied to obtain a LTS for it.

In spite of the interest for MAs, few works addressed their labelled semantics. After early attempts by Cardelli and Gordon (GC03) and (*via* a graphical encoding) by Ferrari, Montanari and Tuosto (FMT01), we are only aware of the papers by Merro and Zappa-Nardelli, collected in (MZN05), and by Rathke and Sobociński (RS08b; RS10).

We already addressed the LTS introduced in (RS08b). We further remark that also Rathke and Sobociński employ a general systematic procedure for deriving LTSs that they previously introduced (RS08a), even if our use of the BC technique automatically guarantees that the strong bisimilarity on the derived LTS is a congruence. With respect to (RS08b), (RS10) presents a purely SOS presentation for the LTS, avoiding the use of the structural congruence in deriving the transition relation. The solution boils down to add to the previous set of rules some symmetric counterparts, dealing with the commutativity of the parallell operator (compare e.g. (RS10, Figure 2) with (RS08b, Figure 2)). The same technique could be pursued in our approach, at the expenses of a more complex set of inferences rules. We refrained from following this path, since the focus of our paper lies in the distillation of the labelled transitions via the BC mechanism However, we remark that the key Theorem 10.1, stating the correspondence between ours and Rathke and Sobociński's LTS for MAs, would still hold in the novel setting.

The LTS proposed by Merro and Zappa-Nardelli (MZN05) is restricted to *systems*: processes obtained by the parallel composition of ambients. For this reason, our rules In, Open and Out have no counterpart. Instead, the rules InAmb, CoIn and OutAmb exactly correspond to the rules (Enter), (Co-Enter), (Exit) in Table 6 of (MZN05). Moreover, our rule CoOpen roughly corresponds to their (Open). Indeed the former inserts a process into the context $-|\mathit{open}\ n.X_1$, while the latter into $k[-|\mathit{open}\ n.X_1|X_2]$ (also due to their restriction to systems). Differently from our LTS, their labels for the rules (Enter) and (Exit) contain the name of the migrating ambient n, thus requiring two extra rules (Enter Shh) and (Exit Shh) for dealing with the possible restriction of n.

For a practitioner, the main interest of our work lies on the presentation of a succinct LTS for MAs, and the associated set of SOS rules. However, we believe that our work represents a relevant case study for the theory of reactive systems (LM00). As pointed out in the introduction, BC rewriting and bigraphical reactive systems (Mil06) are both instances of this theory. Together with (BGK06), this paper shows that the BCs approach is quite effective in deriving LTS for process calculi, and it seems to confirm the advantage of BCs over graphs with interfaces with respect to bigraphs. In bigraphs, all reduction rules must be ground, and thus also the labels and the arriving states of the derived transitions are so. Instead, rewriting with BCs allows to employ few non ground rules (as shown in this paper) and the resulting transitions have labels and arriving states containing (process and name) variables. This feature was not relevant for calculi such as CCS and π, since the variables in the labels occur "outside" of the arriving state and can be forgotten. Consider e.g. the CCS transition $a.b \xrightarrow{-|\bar{a}.Y} b|Y$ derived from the (non

ground) rule $a.X|\bar{a}.Y \to X|Y$. The behaviour of the process $b|Y$ is equivalent to b: their interaction is basically restricted to processes offering a $\bar{b}$ action, and we can thus avoid to consider Y. In the case of MAs, accounting for non ground states is fundamental, because process variables may occur nested inside ambients in arriving states.

The relevance of this work for the theory of reactive systems is not limited to the above observations. The first author has shown in (Bon08) that in reactive systems the bisimilarity on the derived LTS is usually too strict, while *saturated bisimilarity* (i.e., the bisimilarity over the LTS having all contexts as labels) is often more adequate, as for logic programming, open π-calculus (BKM06) and Petri nets (BM07). The present work provides a further successful test of the above claim. In fact, the standard notion of bisimilarity over our LTS is too strict, since it allows to observe the ability of an ambient to migrate, while it should be unobservable, as pointed out in (MZN05). For this reason, Rathke and Sobociński added two extra-rules to their LTS, while Merro and Zappa Nardelli chose an asymmetric definition of bisimilarity. The latter solution recalls the *semi-saturated bisimulation* (BKM06). Instead of requiring that two bisimilar processes must perform transitions with the same label, semi-saturated bisimulation requires that

$$\text{if } P \xrightarrow{C[-]} P_1 \text{ then } C[Q] \text{ reduces to } Q_1 \text{ and } P_1 \; R \; Q_1.$$

It is worth noting that the second and third points of Definition 3.2 in (MZN05) have this shape (the labels $*.enter_n$ and $*.exit_n$ are related to our contexts $-|n[X_1]$ and $n[-|X_1]$). We made precise this correspondence, introducing the notion of *barbed* bisimilarity for reactive systems (BGM09c), and applying it to MAs. We further sketched a correspondence between barbs and semi-saturation (BGM09b): the detailed presentation of the framework obtained by combining the two proposals is going to appear elsewhere.

References

P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Concurrency, Parallelism, and Distribution*, volume 3 of *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 107–187. World Scientific, 1999.

P. Baldan, H. Ehrig, and B. König. Composition and decomposition of DPO transformations with borrowed context. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozemberg, editors, *Graph Transformation*, volume 4178 of *LNCS*, pages 153–167. Springer, 2006.

F. Bonchi, F. Gadducci, and B. König. Process bisimulation via a graphical encoding. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozemberg, editors, *Graph Transformation*, volume 4178 of *LNCS*, pages 168–183. Springer, 2006.

F. Bonchi, F Gadducci, and G. V. Monreale. Labelled transitions for mobile ambients (as synthesized via a graphical encoding). In T. Hildebrandt and D. Gorla, editors, *Expressiveness in Concurrency*, volume 242(1) of *ENTCS*, pages 73–98. Elsevier, 2009.

F. Bonchi, F Gadducci, and G. V. Monreale. On barbs and labels in reactive systems. In B. Klin and P. Sobociński, editors, *Structural Operational Semantics*, volume 18 of *EPTCS.*, pages 46–61, 2009.

F. Bonchi, F Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In L. de Alfaro, editor, *Foundations of Software Science and Computation Structures*, volume 5504 of *LNCS*, pages 272–287. Springer, 2009.

F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *Logic in Computer Science*, pages 69–80. IEEE Computer Society, 2006.

F. Bonchi and U. Montanari. Coalgebraic models for reactive systems. In L. Caires and V.T. Vasconcelos, editors, *Concurrency Theory*, volume 4703 of *LNCS*, pages 364–379. Springer, 2007.

F. Bonchi. *Abstract Semantics by Observable Contexts*. PhD thesis, Department of Informatics, University of Pisa, 2008.

A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Appl. Categ. Struct.*, 7(4):299–331, 1999.

L. Cardelli and A. Gordon. Mobile ambients. *Theor. Comp. Sci.*, 240(1):177–213, 2000.

H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *Math. Struct. in Comp. Sci.*, 16(6):1133–1163, 2006.

G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In A. Restivo, S. Ronchi Della Rocca, and L. Roversi, editors, *Italian Conference on Theoretical Computer Science*, volume 2202 of *LNCS*, pages 1–16. Springer, 2001.

A.D. Gordon and L. Cardelli. Equational properties of mobile ambients. *Math. Struct. in Comp. Sci.*, 13(3):371–408, 2003.

F. Gadducci and U. Montanari. Observing reductions in nominal calculi *via* a graphical encoding of processes. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R.C. de Vrijer, editors, *Processes, terms and cycles (Klop Festschrift)*, volume 3838 of *LNCS*, pages 106–126. Springer, 2005.

F. Gadducci and G. V. Monreale. A decentralized implementation of mobile ambients. In R. Heckel and G. Taentzer, editors, *Graph Transformation*, volume 5214 of *LNCS*, pages 115–130. Springer, 2008.

J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Concurrency Theory*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.

S. Lack and P. Sobociǹski. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(3):511–545, 2005.

R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

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

R. Milner. Pure bigraphs: Structure and dynamics. *Inf. and Comp.*, 204(1):60–122, 2006.

M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of ACM*, 52(6):961–1023, 2005.

J. Rathke and P. Sobociński. Deconstructing behavioural theories of mobility. In G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, editors, *IFIP International Conference on Theoretical Computer Science*, volume 273 of *IFIP*, pages 507–520. Springer, 2008.

J. Rathke and P. Sobociński. Deriving structural labelled transitions for mobile ambients. In F. van Breugel and M. Chechik, editors, *Concurrency Theory*, volume 5201 of *LNCS*, pages 462–476. Springer, 2008.

J. Rathke and P. Sobocinski. Deriving structural labelled transitions for mobile ambients. *Inf. and Comp.*, 208(10):1221–1242, 2010.

P. Sobociński. *Deriving bisimulation congruences from reduction systems*. PhD thesis, BRICS, Department of Computer Science, University of Aarhus, 2004.

V. Sassone and P. Sobociǹski. Reactive systems over cospans. In *Logic in Computer Science*, pages 311–320. IEEE Computer Society, 2005.

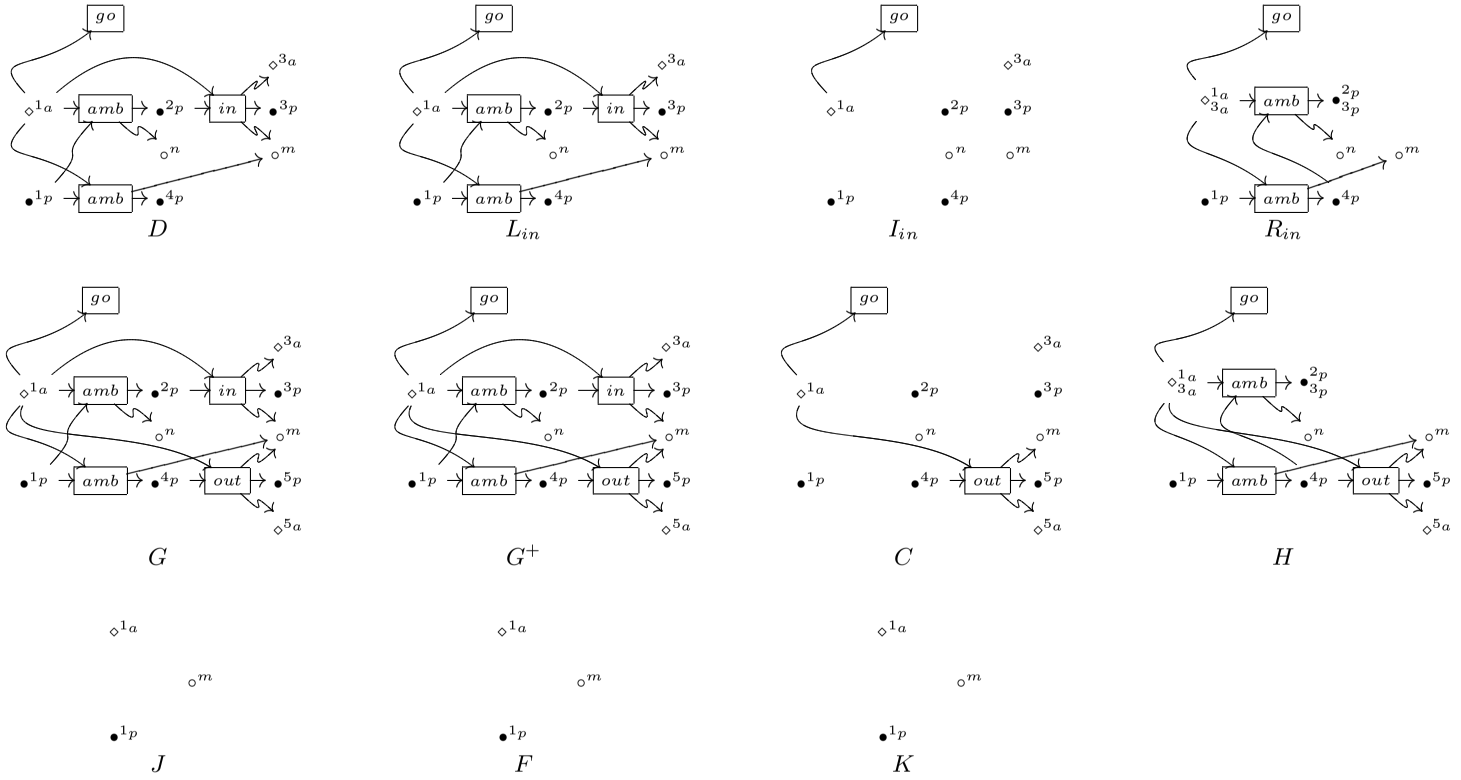


Fig. 20. Ambient n enters ambient m. This corresponds to the transition $(\nu n)(n[in\ m.0]|m[out\ m.0]) \xrightarrow{\cdot} (\nu n)(m[n[0]|out\ m.0])$.

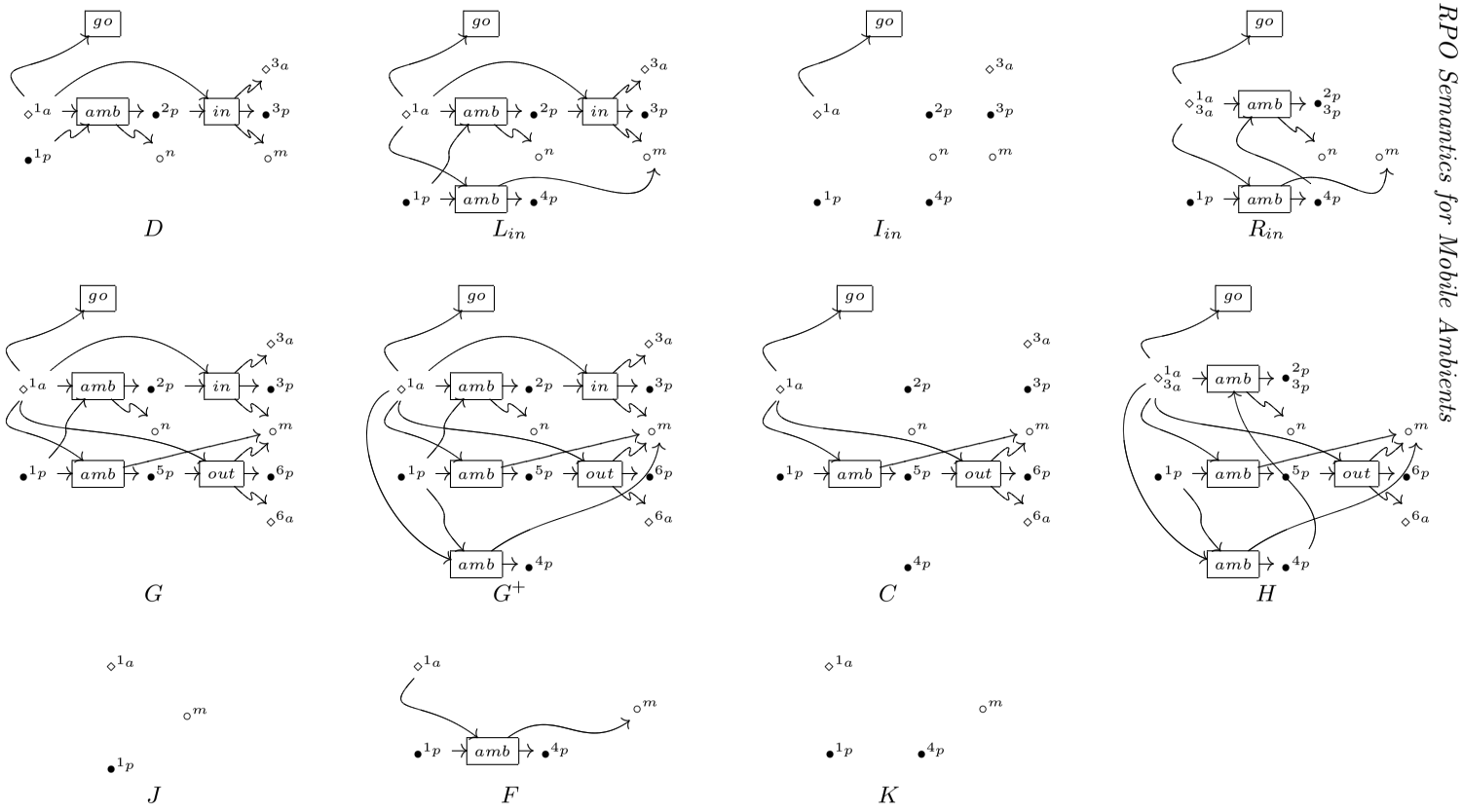


Fig. 21. Ambient n enters ambient m (from environment). This corresponds to the transition $(\nu n)(n[in\ m.0]|m[out\ m.0]) \xrightarrow{-|m[X]} (\nu n)(m[out\ m.0]|m[n[0]|X])$.

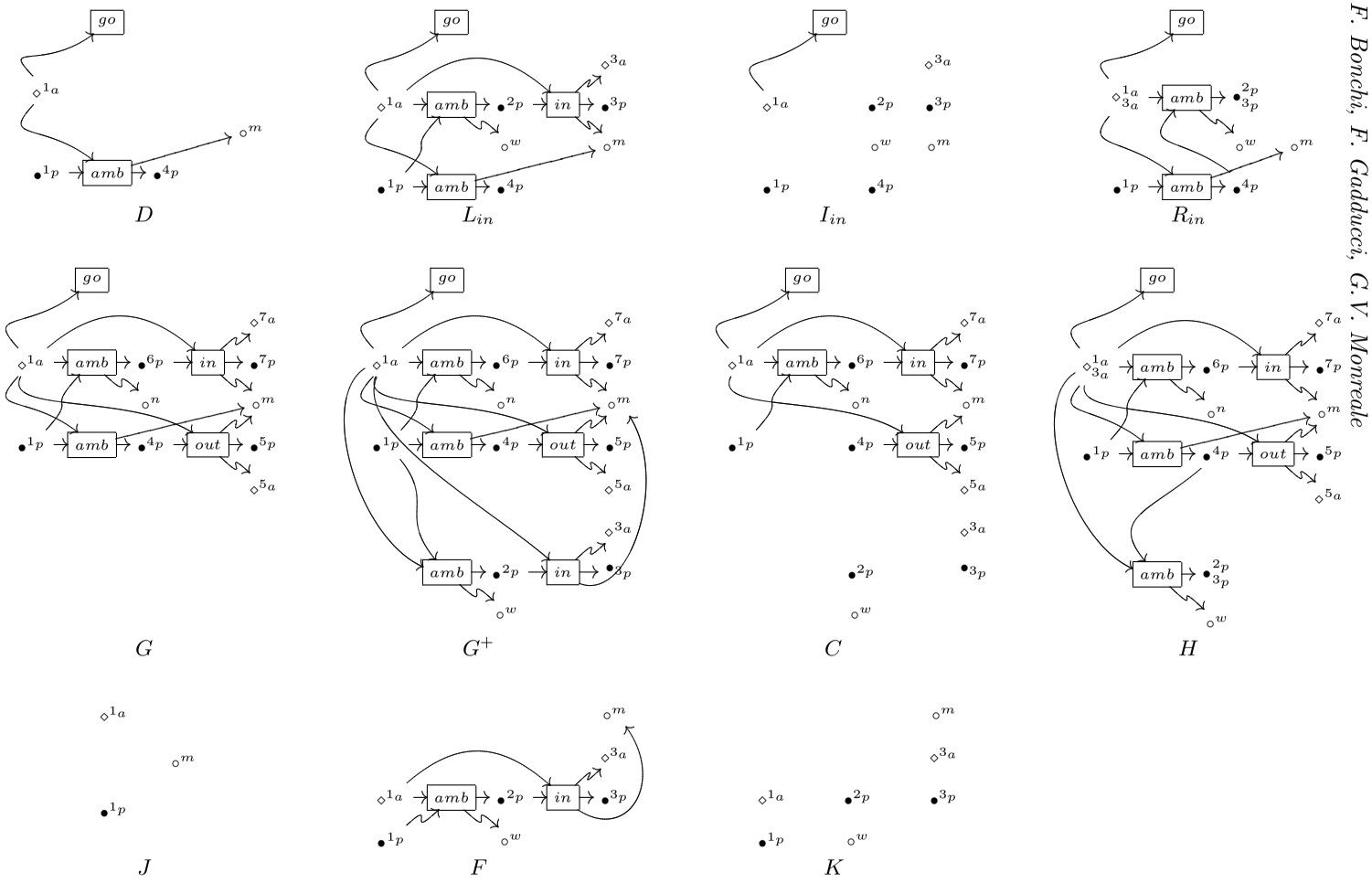


Fig. 22. Ambient w (from environment) enters ambient m. This corresponds to the transition $(\nu n)(n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}]) \xrightarrow{-|w[in\ m.X_2|X_1]} (\nu n)(n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}|w[X_2|X_1]])$.

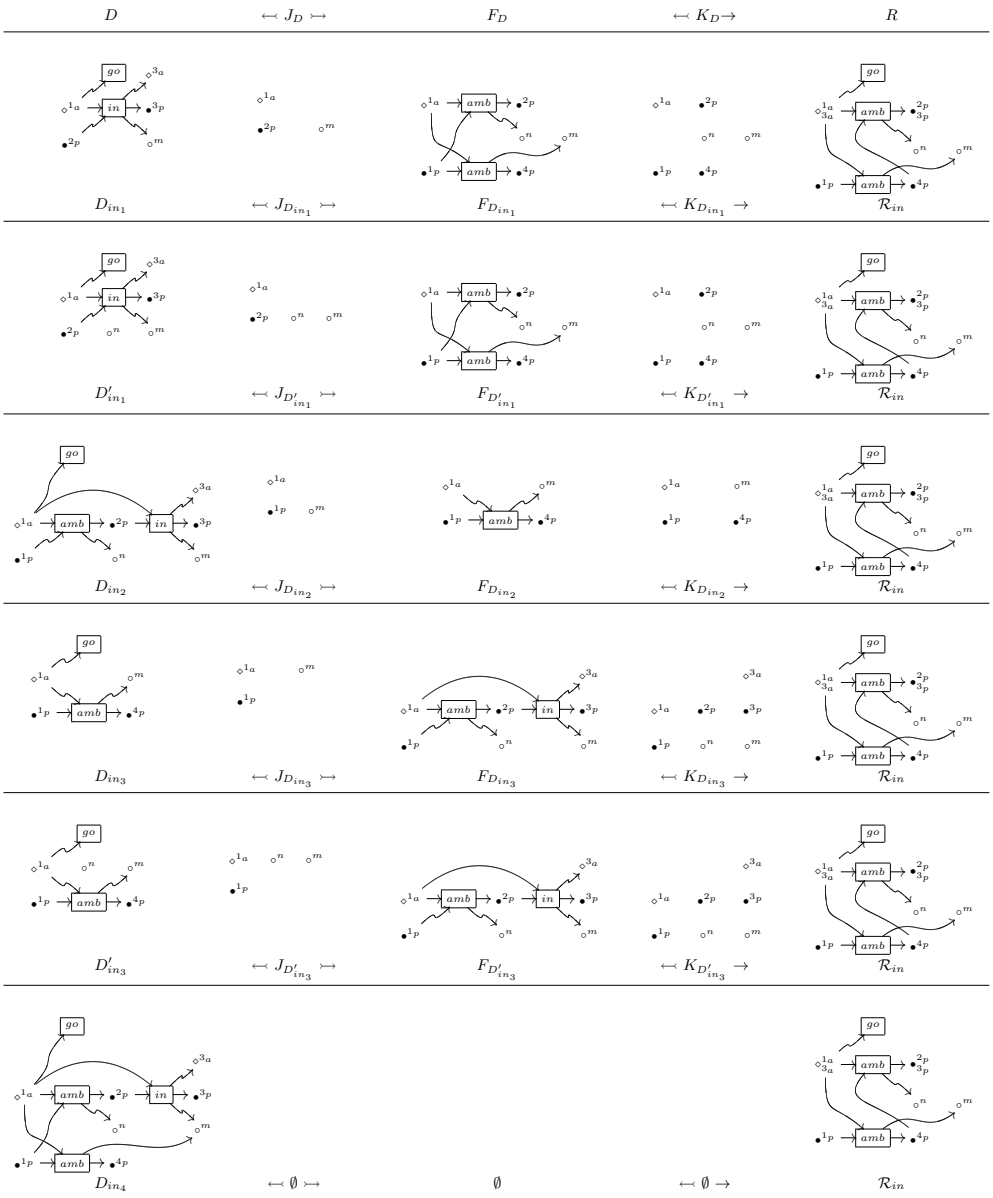


Fig. 23. The minimal transitions generated by the rule p_{in}.

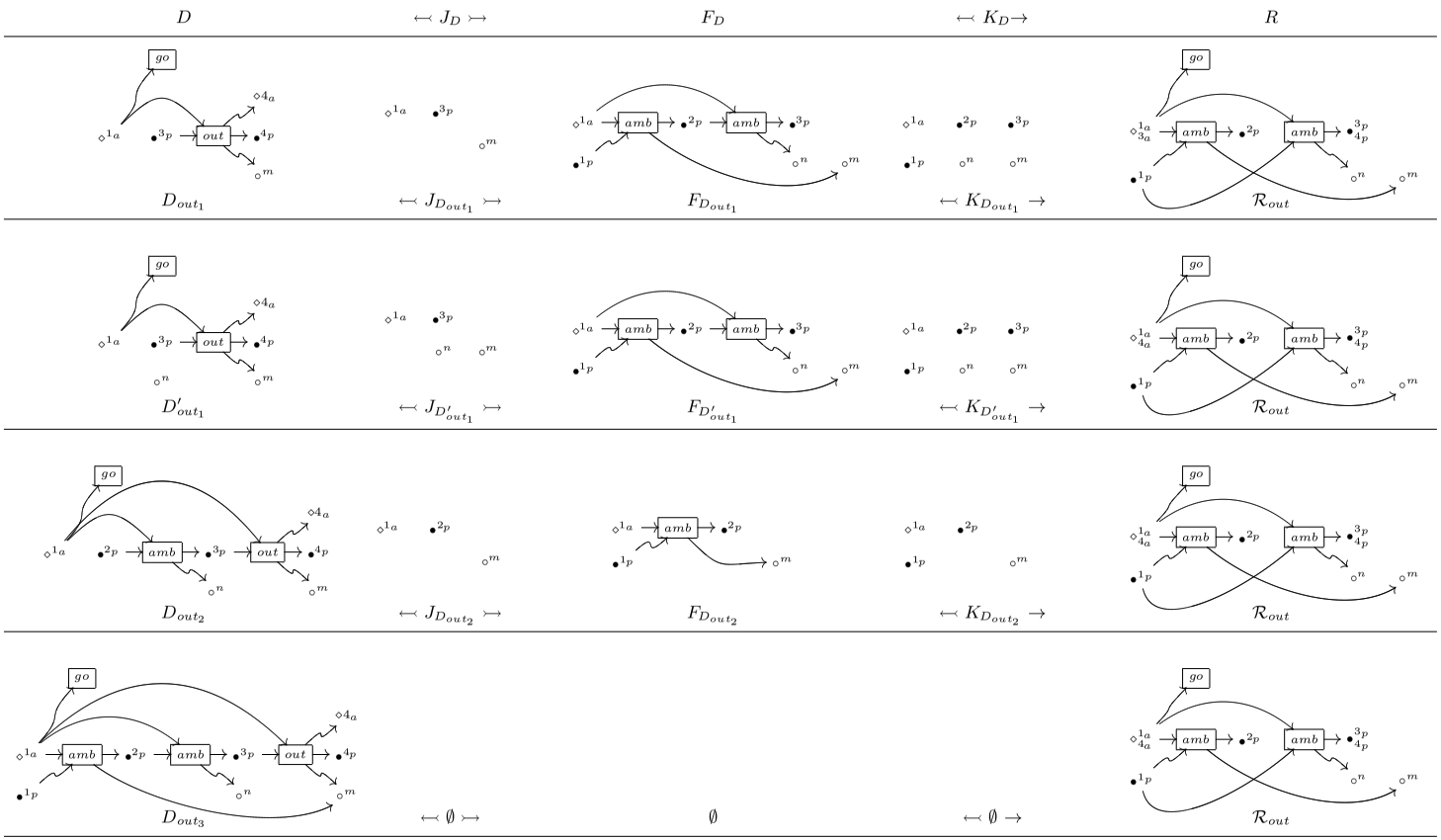


Fig. 24. The minimal transitions generated by the rule p_{out}.

Proofs of Chapter 4

Appendix A. Equivalence between the LTS $\mathcal{D}_{\mathcal{I}}$ and the LTS $\mathcal{S}_{\mathcal{I}}$

This section discusses the equivalence between the LTS $\mathcal{D}_{\mathcal{I}}$, presented in Section 8, and the LTS $\mathcal{S}_{\mathcal{I}}$, introduced in Section 9. In particular, we provide a proof of Theorem 9.1, and to this end, in the following we introduce two useful propositions.

Proposition A.1. Let P be a pure process. If $P \xrightarrow{C[-]_\epsilon}_{\mathcal{D}} Q_\epsilon$ then there exists a linear process Q'_ϵ such that $P \xrightarrow{C[-]_\epsilon}_{\mathcal{S}} Q'_\epsilon$ and for each substitution σ, $Q_\epsilon\sigma \equiv Q'_\epsilon\sigma$.

Sketch The proof is by cases on the rules to obtain $P \xrightarrow{C[-]_\epsilon}_{\mathcal{D}} Q_\epsilon$.

We begin by observing that the rules in Fig. 17 and the rules in the first two rows of Fig. 19 exactly derive the same transition relation of the reduction relation of mobile ambients. So for them the proposition trivially holds.

For the rules in Fig. 18 we show as an example the case of the In rule.

Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{D}} Q_\epsilon$ by In rule. It means that $P \equiv (\nu A)(in\ m.P_1|P_2)$, $m \notin A$, $Q_\epsilon = (\nu A)(m[x[P_1|P_2|X_1]|X_2])$ and $C_\epsilon[-] = x[-|X_1]|m[X_2]$.

We can note that, by applying In rule, $in\ m.P_1 \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{S}} m[x[P_1|X_1]|X_2]$. So, we can apply InPar rule and obtain $in\ m.P_1|P_2 \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{S}} m[x[P_1|P_2|X_1]|X_2]$. Since we also know $m \notin A$, thanks to InRes rule, we can conclude $(\nu A)(in\ m.P_1|P_2) \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{S}} Q_\epsilon$, therefore $P \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{S}} Q_\epsilon$ and trivially, for each substitution σ, $Q_\epsilon\sigma \equiv Q_\epsilon\sigma$. □

Proposition A.2. Let P be a pure process. If $P \xrightarrow{C[-]_\epsilon}_{\mathcal{S}} Q_\epsilon$ then there exists a linear process Q'_ϵ such that $P \xrightarrow{C[-]_\epsilon}_{\mathcal{D}} Q'_\epsilon$ and for each substitution σ, $Q_\epsilon\sigma \equiv Q'_\epsilon\sigma$.

Sketch The proof is by cases on the rules to obtain $P \xrightarrow{C[-]_\epsilon}_{\mathcal{S}} Q_\epsilon$.

As in the proof above, for the rules in the first two rows of Fig. 19 the proposition trivially holds. Instead, for the remaining rules of the same figure, we show as an example the cases for In and InPar rules.

— Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ by In rule. It means that $P \equiv in\ m.P_1$, $Q_\epsilon = m[x[P_1|X_1]|X_2]$ and $C_\epsilon[-] = x[-|X_1]|m[X_2]$. It is easy to check that $P \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{D}} Q_\epsilon$ by In rule, so the proposition trivially holds.

— Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ by InPar rule. This means that $P \equiv P'|R'$, $C_\epsilon[-] = x[-|X_1]|m[X_2]$, $P' \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q'_\epsilon$ and $Q_\epsilon = Q''_\epsilon\{^{R'|X_1}/_{X_1}\}$.
By induction hypothesis, we have $P' \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{D}} Q''_\epsilon$. This means that $P' \equiv (\nu A)(in\ m.P_1|P_2)$, $m \notin A$ and $Q''_\epsilon = (\nu A)(m[x[P_1|P_2|X_1]|X_2])$. Note that $P'|R' \equiv (\nu A)(in\ m.P_1|P_2)|R'$ and $(\nu A)(in\ m.P_1|P_2)|R' \equiv (\nu A')(in\ m.P'_1|P'_2|R')$, by considering $(\nu A)(in\ m.P_1|P_2)$ α-equivalent to $(\nu A')(in\ m.P'_1|P'_2)$ and $A' \cap fn(R') = \emptyset$. So, thanks to In rule, $P'|R' \xrightarrow{x[-|X_1]|m[X_2]}_{\mathcal{D}} Q'_\epsilon$, where $Q'_\epsilon = (\nu A')(m[x[P'_1|P'_2|R'|X_1]|X_2])$ and it is easy to check that for each substitution σ, $Q_\epsilon\sigma \equiv Q'_\epsilon\sigma$

□

Theorem 9.1 trivially follows from the two propositions above and from Definitions 8.1 and 9.1.

Appendix B. Correspondence between the LTS $\mathcal{S}_\mathcal{I}$ and the LTS $\mathcal{CA}$

This section shows the proofs of Propositions 10.1 and 10.2 used to formally prove the correspondence between our LTS $\mathcal{S}_I$, defined on pure processes of mobile ambients, and the LTS $\mathcal{CA}$ for mobile ambients proposed by Rathke and Sobociński in (RS08b).

First of all, we introduce the proof of Proposition 10.1, needed to prove the first statement of Theorem 10.1.

Proof sketch of Proposition 10.1 The proof is by cases on the rules to obtain $P \xrightarrow{\alpha\downarrow\vec{M}^\alpha}_{\mathcal{CA}} Q$. We only show the proof for some rules, because the other cases are analogous.

— Assume that $P \xrightarrow{\alpha\downarrow\vec{M}^\alpha}_{\mathcal{CA}} Q$ by $C\lambda$ rule. It means that $P \xrightarrow{\alpha}_{\mathcal{C}} A$, $A \xrightarrow{\vec{M}^\alpha\downarrow}_{\mathcal{A}} Q$ and $\alpha \notin \{[\overline{in\, m}], \overline{open\, n}, \tau\}$. Now we proceed by cases on the rules to obtain $P \xrightarrow{\alpha}_{\mathcal{C}} A$ with $\alpha \notin \{[\overline{in\, m}], \overline{open\, n}, \tau\}$. As an example, we show the cases of the In and ||In rules.
Assume that $P \xrightarrow{\alpha}_{\mathcal{C}} A$ by In rule. It means that $P \equiv in\ m.P_1$, $A = \lambda X x Y.m[x[P_1|X]|Y]$ and $\alpha = in\ m$. We assume that $\vec{M}^\alpha = R, n, S$, for R, S processes and n ambient name, therefore we have $Q \equiv m[n[P_1|R]|S]$. We have to show that there exists Q_ϵ, such that $P \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} Q_\epsilon$ and $Q \equiv Q_\epsilon \sigma_M^\alpha$, with $\sigma_M^\alpha = \{^R/_{X_1}, ^n/_x, ^S/_{X_2}\}$. We take $Q_\epsilon = m[x[P_1|X_1]|X_2]$. It is easy to check that $in\ m.P_1 \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} m[x[P_1|X_1]|X_2]$ by In rule. Moreover, we have $m[x[P_1|X_1]|X_2]\sigma_M^\alpha = m[n[P_1|R]|S] \equiv Q$.
Assume that $P \xrightarrow{\alpha}_{\mathcal{C}} A$ by ||In rule. It means that $\alpha = in\ m$, $P \equiv P_1|P_2$, $P_1 \xrightarrow{in\ m}_{\mathcal{C}} A'$, and $A = \lambda X.A'(P_2|X)$. We assume that $\vec{M}^\alpha = R, n, S$, for R, S processes and n ambient name, therefore we have $Q \equiv A'(P_2|R, n, S)$. Let us consider $\vec{M}'^\alpha = P_2|R, n, S$. Since $A \xrightarrow{\vec{M}^\alpha\downarrow}_{\mathcal{A}} Q$, then $A' \xrightarrow{\vec{M}'^\alpha\downarrow}_{\mathcal{A}} Q$. Therefore, we have $P_1 \xrightarrow{in\ m\downarrow\vec{M}'^\alpha}_{\mathcal{CA}} Q$. By induction hypothesis, there exists Q'_ϵ such that $P_1 \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} Q'_\epsilon$ and $Q \equiv Q'_\epsilon \sigma_{M'}^\alpha$, where $\sigma_{M'}^\alpha = \{^{P_2|R}/_{X_1}, ^n/_x, ^S/_{X_2}\}$. We have to show that there exists Q_ϵ, such that $P \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} Q_\epsilon$ and $Q \equiv Q_\epsilon \sigma_M^\alpha$, with $\sigma_M^\alpha = \{^R/_{X_1}, ^n/_x, ^S/_{X_2}\}$. We take $Q_\epsilon = Q'_\epsilon\{^{P_2|X_1}/_{X_1}\}$. It is easy to check that $P_1|P_2 \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} Q'_\epsilon\{^{P_2|X_1}/_{X_1}\}$ by InPar rule. Moreover, it is obvious that $Q'_\epsilon\{^{P_2|X_1}/_{X_1}\}\{^R/_{X_1}, ^n/_x, ^S/_{X_2}\} = Q'_\epsilon\{^{P_2|R}/_{X_1}, ^n/_x, ^S/_{X_2}\} \equiv Q$.

— Assume that $P \xrightarrow{\alpha\downarrow\vec{M}^\alpha}_{\mathcal{CA}} Q$ by coInλ rule. This means that $\alpha = [\overline{in\, m}]$, $P \xrightarrow{[\overline{in\, m}]}_{\mathcal{C}} A$, $\vec{M}^\alpha = R, S, n$ and $A(\lambda XYZx.m[x[Y|Z]|X]) \xrightarrow{R,S,n\downarrow}_{\mathcal{A}} Q$. Now we proceed by cases on the rules to obtain $P \xrightarrow{[\overline{in\, m}]}_{\mathcal{C}} A$. As an example, we show the case of the coIn rule.
Assume that $P \xrightarrow{[\overline{in\, m}]}_{\mathcal{C}} A$ by coIn rule. It means that $P \equiv m[P_1]$ and $A = \lambda Z.Z(P_1)$, and hence we have $Q \equiv m[n[R|S]|P_1]$. We have to show that there exists Q_ϵ, such that $P \xrightarrow{C_\epsilon^{[\overline{in\, m}]}[-]}_{\mathcal{S}} Q_\epsilon$ and $Q \equiv Q_\epsilon \sigma_M^\alpha$, with $\sigma_M^\alpha = \{^R/_{X_1}, ^S/_{X_2}, ^n/_x\}$. We take $Q_\epsilon = m[x[X_1|X_2]|P_1]$. It is easy to check that $m[P_1] \xrightarrow{C_\epsilon^{[\overline{in\, m}]}[-]}_{\mathcal{S}} m[x[X_1|X_2]|P_1]$ by CoIn rule. and $m[x[X_1|X_2]|P_1]\{^R/_{X_1}, ^S/_{X_2}, ^n/_x\} = m[n[R|S]|P_1] \equiv Q$.

□

Now we show the proof of Proposition 10.2, needed to prove the second statement of Theorem 10.1.

Proof sketch of Proposition 10.2 The proof proceeds by cases on the rules to obtain $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$. We only show some cases, because the other ones are analogous.

— Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ by In rule. This means that $\alpha = in\ m$, $P \equiv in\ m.P_1$, $C_\epsilon^{in\ m}[-] = x[-|X_1]|m[X_2]$ and $Q_\epsilon = m[x[P_1|X_1]|X_2]$. Moreover, the substitution σ has the following shape $\{^{P_2}/_{X_1}, ^{n}/_{x}, ^{P_3}/_{X_2}\}$, for some ambient name n and some processes P_1 and P_2. Therefore, we have $Q \equiv Q_\epsilon\sigma = m[x[P_1|X_1]|X_2]\{^{P_2}/_{X_1}, ^{n}/_{x}, ^{P_3}/_{X_2}\} = m[n[P_1|P_2]|P_3]$. We have to show that $P \xrightarrow{in\ m \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q$, where $\vec{M}_\sigma^\alpha = P_2, n, P_3$. It is easy to check that $P \xrightarrow{in\ m}_{\mathcal{C}} \lambda XxY.m[x[P_1|X]|Y]$ tanks to In rule in Figure 6 of (RS08b). Moreover, we can apply Inst rule shown in Figure 7 of (RS08b), and say $\lambda XxY.m[x[P_1|X]|Y] \xrightarrow{\vec{M}_\sigma^\alpha \downarrow}_{\mathcal{A}} m[n[P_1|P_2]|P_3]$. Therefore, thanks to C$\lambda$ rule in Figure 8 of (RS08b), we can conclude $P \xrightarrow{in\ m \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q$.

— Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ by InPar rule. This means that $\alpha = in\ m$, $P \equiv P_1|Q_1$, $C_\epsilon^{in\ m}[-] = x[-|X_1]|m[X_2]$, $P_1 \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} P_\epsilon$ and $Q_\epsilon = P_\epsilon\{^{Q_1|X_1}/_{X_1}\}$. Moreover, the substitution σ has the following shape $\{^{P_2}/_{X_1}, ^{n}/_{x}, ^{P_3}/_{X_2}\}$, for some ambient name n and some processes P_1 and P_2. Consider the substitution $\sigma' = \{^{Q_1|P_2}/_{X_1}, ^{n}/_{x}, ^{P_3}/_{X_2}\}$. Note that $P_\epsilon\sigma' = Q_\epsilon\sigma \equiv Q$. Since $P_1 \xrightarrow{C_\epsilon^{in\ m}[-]}_{\mathcal{S}} P_\epsilon$, then, by applying the induction hypothesis, we have $P_1 \xrightarrow{in\ m \downarrow \vec{M}_{\sigma'}^\alpha}_{\mathcal{CA}} P_\epsilon\sigma'$, where $\vec{M}_{\sigma'}^\alpha = Q_1|P_2, n, P_3$. We have to show that $P \xrightarrow{in\ m \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q_\epsilon\sigma$, where $\vec{M}_\sigma^\alpha = P_2, n, P_3$. We know that $P_1 \xrightarrow{in\ m \downarrow \vec{M}_{\sigma'}^\alpha}_{\mathcal{CA}} P_\epsilon\sigma'$. This means that $P_1 \xrightarrow{in\ m}_{\mathcal{C}} A$ and $A \xrightarrow{\vec{M}_{\sigma'}^\alpha \downarrow}_{\mathcal{A}} P_\epsilon\sigma'$. Since $P_1 \xrightarrow{in\ m}_{\mathcal{C}} A$, thanks to ||In rule of Figure 6 in (RS08b), we have $P_1|Q_1 \xrightarrow{in\ m}_{\mathcal{C}} \lambda X.A(Q_1|X)$. It is easy to check that if $A \xrightarrow{\vec{M}_{\sigma'}^\alpha \downarrow}_{\mathcal{A}} P_\epsilon\sigma'$, then we also have $\lambda X.A(Q_1|X) \xrightarrow{\vec{M}_\sigma^\alpha \downarrow}_{\mathcal{A}} P_\epsilon\sigma'$. Therefore, by applying Cλ rule in Figure 8 of (RS08b), we obtain $P_1|Q_1 \xrightarrow{in\ m \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} P_\epsilon\sigma'$ and so $P \xrightarrow{in\ m \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q$.

— Assume that $P \xrightarrow{C_\epsilon[-]}_{\mathcal{S}} Q_\epsilon$ by CoIn rule. This means that $\alpha = \overline{[in\ m]}$, $P \equiv m[P_1]$, $C_\epsilon^{\overline{[in\ m]}}[-] = -|x[in\ m.X_1|X_2]$ and $Q_\epsilon = m[x[X_1|X_2]|P_1]$. Moreover, the substitution σ has the following shape $\{^{P_2}/_{X_1}, ^{P_3}/_{X_2}, ^{n}/_{x}\}$, for some ambient name n and some processes P_1 and P_2. So, we have $Q \equiv Q_\epsilon\sigma = m[n[P_2|P_3]|P_1]$. We have to show that $P \xrightarrow{\overline{[in\ m]} \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q_\epsilon\sigma$, where $\vec{M}_\sigma^\alpha = P_2, P_3, n$. It is easy to check that $P \xrightarrow{\overline{[in\ m]}}_{\mathcal{C}} \lambda Z.Z(P_1)$, by CoIn rule in Figure 6 of (RS08b). Moreover, by Inst rule shown in Figure 7 of (RS08b), $(\lambda Z.Z(P_1))(\lambda XYZx.m[x[Y|Z]|X]) \xrightarrow{\vec{M}_\sigma^\alpha \downarrow}_{\mathcal{A}} m[n[P_2|P_3]|P_1]$. Therefore, thanks to CoInλ rule of Figure 8 of (RS08b), we can conclude $P \xrightarrow{\overline{[in\ m]} \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} m[n[P_2|P_3]|P_1]$, and so $P \xrightarrow{\overline{[in\ m]} \downarrow \vec{M}_\sigma^\alpha}_{\mathcal{CA}} Q$.

□