Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 242 (2009) 73–98

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Labelled Transitions for Mobile Ambients (As Synthesized via a Graphical Encoding)

Filippo Bonchi, Fabio Gadducci, Giacoma Valentina Monreale

Dipartimento di Informatica, Università di Pisa
largo Pontecorvo 3c, I-56127 Pisa, Italy
gadducci@di.unipi.it, fibonchi@di.unipi.it, vale@di.unipi.it

Abstract

The paper presents a case study on the synthesis of labelled transition systems (LTSs) for process calculi, choosing as testbed Cardelli and Gordon's Mobile Ambients (MAs). The proposal is based on a graphical encoding: each process is mapped into a graph equipped with suitable interfaces, such that the denotation is fully abstract with respect to the usual structural congruence. Graphs with interfaces are amenable to the synthesis mechanism proposed by Ehrig and König and based on borrowed contexts (BCs), an instance of relative pushouts, introduced by Leifer and Milner. The BC mechanism allows the effective construction of a LTS that has graphs with interfaces as both states and labels, and such that the associated bisimilarity is automatically a congruence. Our paper focuses on the analysis of a LTS over (processes as) graphs with interfaces, as distilled by exploiting the graphical encoding of MAs. In particular, we use the LTS on graphs to recover a suitable LTS directly defined over the structure of MAs processes.

Keywords: Labelled transition system, mobile ambients, borrowed contexts

1 Introduction

Among recently introduced process calculi, mobile ambients [7] (MAs) possibly represents the most fruitful proposal so far. The analogy between ambients and network domains, explicitly addressed since the beginning, and between ambients and molecular environments, often exploited in system biology [21], made MAs a centerpiece in recent applications and development of the process calculi paradigm.

It is then baffling that the calculus has been so resilient to the introduction of an observational semantics, based on a labelled transition system (LTS). Indeed, after Milner's treatment of π-calculus [18], it is now customary to present the semantics of a calculus with a reduction semantics, modulo a congruence equating those processes which intuitively represent the same distributed system. As for the case of MAs, the set of rules defining the original reduction semantics is rather complex. Indeed,

[1] Research partially supported by the IST 2004-16004 SEnSOria.

doi:10.1016/j.entcs.2009.06.014

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 presence of processes Q and R, which denote the context into which the actual instance of the rule has to be mapped into. In general terms, the need of such a rich contextual information makes more difficult to obtain a satisfying observational semantics. After the initial attempt by Cardelli and Gordon [14], and an early proposal by Ferrari, Montanari and Tuosto [11] exploiting a graphical encoding of processes, we are aware of the work by Merro and Zappa-Nardelli [16] and, quite recently, by Rathke and Sobociński [23].

A series of papers recently addressed the need of synthesizing a LTS out of the reduction semantics of e.g. a calculus. The most successful technique so far has been proposed by Leifer and Milner with the so-called *relative pushout* (RPO) [15], which captures in an abstract setting the intuitive notion of minimal context into which a process has to be inserted, in order for allowing a reduction to occur.

However, proving that a calculus satisfies the requirements needed for applying the RPOs technique is often quite a daunting task, due to the intricacies of the structural congruence. A way out of the impasse is represented by looking for graphical encodings of processes, such that process congruence is turned into graph isomorphism. Graphs are amenable to the RPOs trappings, and once the processes of a calculus have been encoded as graphs, a suitable LTS can be distilled. Indeed, the main source of examples concerning RPOs have been *bigraphs* [19], a graphical formalism introduced by Milner for specifying concurrent and distributed systems.

It is noteworthy that, should the reduction relation over graphs be defined using the double pushout (DPO) approach [1], these graphs are amenable to the *borrowed context* (BC) technique, developed by Ehrig and König, which offers an algorithmic solution for calculating the minimal contexts enabling a graph transformation rule

So, the approach pursued in this and other papers [4,13] is quite straightforward: for a given calculus, a graphical encoding (over standard graphs) is found such that process 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 is thus immediately distilled. Indeed, 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 admittedly one of simplest calculus available, namely, Milner's CCS [17].

In this paper we exploit the graphical encoding for MAs, proposed in [12], to distill a LTS on (processes encoded as) graphs. This LTS is then used to infer a set of rules defined on the processes of the MAs calculus, and we compare it with the alternative solutions proposed so far, discovering many similarities (thus confirming the hints provided by the ingenuity of the researchers), yet with a few substantial differences, as articulated in the concluding section. Since we are interested in LTS defined over processes, we provide a comparison with the only two works presenting a LTS on MAs processes, namely, those proposed in [16,23].

This paper is organized as follows. Section 2 briefly recalls the MAs calculus. In Section 3 we introduce (typed hyper-)graphs and their extension with interfaces, while Section 4 presents DPO rewriting on graphs with interfaces as well as the BC technique for distilling a LTS. Then, in Section 5 we recall a graphical encoding for MAs processes that has been introduced in [12]. A graph transformation system for MAs that simulates process reduction is defined in Section 6. Section 7 presents a LTS for graphs representing MAs processes, obtained by means of the BC synthesis mechanism. Section 8 introduces a LTS defined over processes of the MAs calculus and obtained from the LTS over graphs. Finally, Section 9 concludes the paper.

2 Mobile Ambients

This section shortly recalls the finite, communication-free fragment of mobile ambients [7], its structural equivalence and the associated reduction semantics.

Table 1 shows the syntax of the calculus. We assume a set $\mathcal{N}$ of *names* ranged over by $m, n, u, \ldots$. Besides the standard constructors, we included a set of *process variables* $\mathcal{X} = \{X, Y, \ldots\}$, and a set of *name variables* $\mathcal{V} = \{x, y, \ldots\}$. 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. These are needed for the presentation of the LTS in Section 8.

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

Table 1
(Extended) Syntax of mobile ambients.

We use the standard definitions for the set of free names of a process P, denoted by $fn(P)$, and for α-convertibility, with respect to the restriction operators (νn). We let $P, Q, R, \ldots$ range over the set $\mathcal{P}$ of *pure* processes, i.e., such that neither process nor name variable is contained. While $P_\epsilon, Q_\epsilon, R_\epsilon, \ldots$ range over the set $\mathcal{P}_\epsilon$ of *well-formed* processes, i.e., such that no process or ambient variable occurs twice.

We also 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 $(\nu n)(\nu m)(X|x[\mathtt{0}])\{^{m}/_{x}, ^{n[P]}/_{X}\}$ corresponds to the pure process $(\nu p)(\nu q)(n[P]|m[\mathtt{0}])$, for names $p, q \notin \{m\} \cup fnn[P]$.

The semantics of the calculus is given by means of a reduction relation and a *structural congruence*, denoted by $\equiv$, which is the least equivalence on pure processes that satisfies the equations and the rules shown 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 the operational semantics.

The *reduction relation*, denoted by $\rightarrow$, describes the evolution of processes over time: $P \rightarrow Q$ means that P reduces to Q, that is, P can execute a computational step and it is transformed into Q. Table 3 shows the reduction rules.

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

Table 2
Structural congruence on pure processes.

$n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
$m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$
$open\ n.P\|n[Q] \rightarrow P\|Q$	$P \rightarrow Q \Rightarrow P\|R \rightarrow Q\|R$

Table 3
Reduction relation on pure processes.

The reduction relation is closed with respect to structural congruence. Note that our chosen congruence slightly differs from the standard one, since we drop the axiom $(\nu n)\mathsf{0} \equiv \mathsf{0}$, 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. Indeed, the equality induced by the latter axiom holds in the observational equivalence proposed by Merro and Zappa Nardelli [16]. In particular, two processes that are structurally congruent according to the axiom *Cong-Res-Act* are *reduction barbed* congruent.

3 Graphs and Their Extension with Interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [8] for a more detailed introduction.

Definition 3.1 (graphs) *A* (hyper-)graph *is a four-tuple* $\langle V, E, s, t\rangle$ *where* V, E *are the sets of nodes and edges and* $s, t : E \rightarrow V^*$ *are the source and target functions. A* graph morphism *is a pair of functions* $\langle f_V, f_E\rangle$ *preserving source and target.*

The corresponding category is denoted by **Graph**. However, we often consider *typed graphs* [9], i.e., graphs labelled over a structure that is itself a graph.

Definition 3.2 (typed graphs) *Let* T *be a graph. A* typed graph G *over* T *is a graph* $|G|$*, together with a graph morphism* $t_G : |G| \rightarrow T$*. A* T-typed graph morphism *is a graph morphism* $f : |G_1| \rightarrow |G_2|$ *preserving the typing.*

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

Definition 3.3 (graphs with interfaces) *Let* J, 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 $f : \mathbb{G} \Rightarrow \mathbb{H}$ *is a typed graph morphism* $f : G \to H$ *between the underlying graphs that preserves the input and output morphisms.*

We let $J \xrightarrow{j} G \xleftarrow{k} K$ denote a graph with interfaces J and K.[2] 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 $J \to G$.

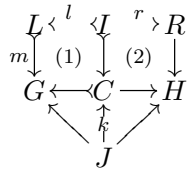
In order to define a process encoding, some (binary) operators on graphs with discrete interfaces should be defined. Since we rely on the encoding presented in [12], we refer the reader there for details, and to Appendix A for a quick survey.

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).

Definition 4.1 (graph production) *A* T*-typed* graph production *is a span* $L \xleftarrow{l} I \xrightarrow{r} R$ *with* l *mono in* T-**Graph**. *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 *assigns each production name to a* T*-typed production.*

Definition 4.2 (derivation of graphs with interfaces) *Let* $J \to G$ *and* $J \to H$ *be two graphs with interfaces. Given a production* $p : L \leftarrowtail I \longrightarrow R$, *a* match *of* p *in* G *is a mono* $m : L \rightarrowtail G$. *A* direct derivation *from* $J \to G$ *to* $J \to H$ *via* p *and* m *is a diagram as depicted in the right, where (1) and (2) are pushouts and the bottom triangles commute. In this case we write* $J \to G \Longrightarrow J \to H$.



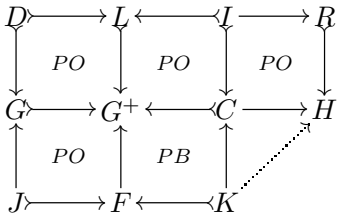
The morphism $k : J \to C$ which makes the left triangle commute is unique, whenever it exists. If such a morphism does not exist, then the rewriting step is not feasible. Moreover, note that the standard DPO derivations can be seen as a special instance of these, obtained considering as interface J the empty graph.

In these derivations, the left-hand side L of a production must occur completely in G. In a *borrowed context* (BC) derivation the graph L might occur partially in G, since the latter may interact with the environment through J in order to exactly match L. Those BCs are the "smallest" extra contexts needed to obtain the image of L in G. The mechanism was introduced in [10] in order to derive a LTS from direct derivations, using BCs as labels. The following definition is lifted from [22], extended by including morphisms that are not necessarily mono.

Definition 4.3 (rewriting with borrowed contexts) *Given a production* $p : L \xleftarrow{l} I \xrightarrow{r} R$, *a graph with interfaces* $J \to G$ *and a mono* $d : D \rightarrowtail L$, *we say*

[2] With an abuse of terminology, we sometimes refer to the image of the input and output morphisms as inputs and outputs, respectively. Thus, in the following we often refer implicitly to a graph with interfaces as the representative of its isomorphism class, still using the same symbols to denote it and its components.

that $J \to G$ reduces to $K \to H$ *with transition label* $J \rightarrowtail F \leftarrowtail K$ via p *and* d *if there are graphs* G^+, C *and additional morphisms such that the diagram below commutes and the squares are either pushouts (PO) or pullbacks (PB). In this case we write* $J \to G \xrightarrow{J \rightarrowtail F \leftarrowtail K} K \to H$, *also called* rewriting step with borrowed context.



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

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. After the description a type graph (T_M, depicted in Figure 1), the encoding is defined by means of suitable operators on typed graphs with interfaces. This corresponds to a variant of the usual 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.

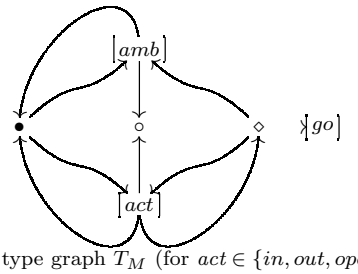


Fig. 1. The type graph T_M (for $act \in \{in, out, open\}$).

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 dependences between the operators of the process and their activation dependences.

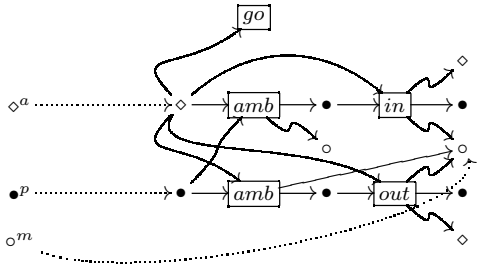


Fig. 2. Graph encoding for the process $(\nu n)(n[in\ m.0]|m[out\ m.0])$.

Each edge of the type graph, except the *go* edge, simulates an operator of MAs. Note that the *act* edge represents three edges, namely *in*, *out* and *open*. These edges simulate the capabilities of the calculus, while the *amb* edge simulates the ambient operator, and there are no edges to simulate the restriction operator and the parallel composition. Finally, the *go* edge is a syntactical device for detecting the "entry" point for the computation. We need it later to simulate MAs reduction semantics. It allows to avoid the occurrence of a reduction underneath a *act* operator.

We remark that choosing a graph typed over T_M means to consider graphs where each node (edge) is labelled by a node (edge) of that type graph, and the incoming and outcoming tentacles are preserved. We refer the reader to [12] for the formal presentation of the encoding, or to Appendix B for a short recollection.

For our purposes it then suffices to say that the encoding $[\![P]\!]^{go}_{\Gamma}$ of a pure process P, where Γ is a set of names such that $fn(P) \subseteq \Gamma$, is a graph with interfaces $(\{a, p\} \cup \Gamma, \emptyset)$, for $a, p \notin \mathcal{N}$. Our encoding is sound and complete with respect to the structural congruence $\equiv$, as stated by the proposition below.

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 $[\![P]\!]^{go}_{\Gamma} = [\![Q]\!]^{go}_{\Gamma}$.*

Example 5.2 Consider the pure process $P = (\nu n)(n[in\ m.0]|m[out\ m.0])$. It is a very simple process, which represents a restricted ambient n that can enter an ambient m. Figures 2 depicts the graph encoding for the process above. The leftmost edges, both labelled *amb*, have the same roots, into which the nodes of the interface a and p are mapped. Those two edges represent the topmost operators of the two parallel components of the process. The edge *in* represents the operator *in* m that is inside the restricted ambient n, while the edge *out* represents the operator *out* m that is inside the ambient m. These two last edges are linked to the same root node $\diamond$ of their parent ambients. Intuitively, this means that they can be involved in a reduction step, too, since the only edge labelled *go* is linked to that same node. Note that the ambient name m is in the interface since it is free in P, instead the name n, which is bound, does not belong to the interface.

6 Graph Transformation for Mobile Ambients

This section presents a graph transformation system (GTS) that models the reduction semantics of the MAs calculus.

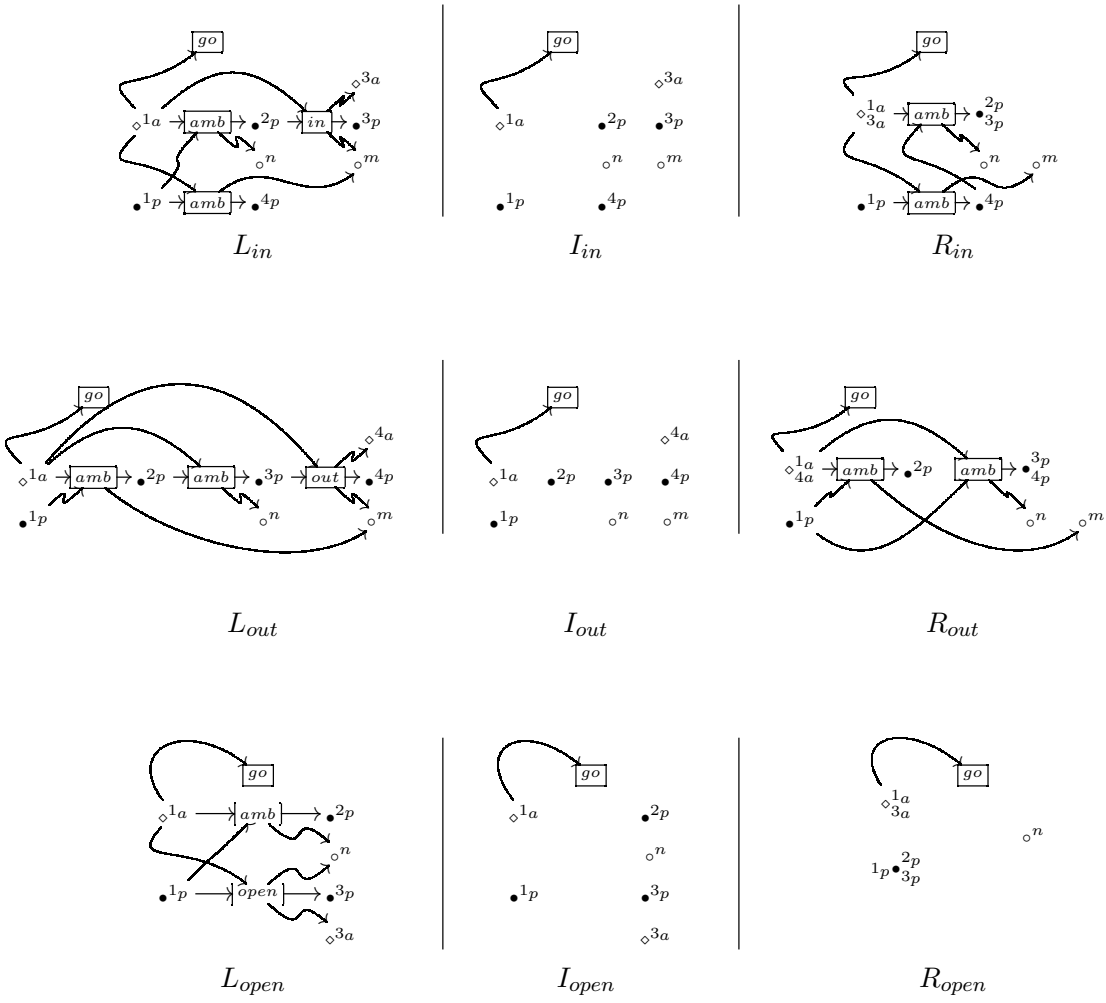


Fig. 3. The rewriting rules p_{in}, p_{out} and p_{open}.

Figure 3 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, namely p_{in}, p_{out} and p_{open}. They simulate the three axioms of the reductions relation. The action of the three rules is described by the node identifiers. These identifiers are of course arbitrary: they correspond to the actual elements of the set of nodes and are just used to characterize the span of functions.

It seems noteworthy that three rules[3] suffice for recasting the reduction semantics of mobile ambients. That is possible for two reasons. First, the closure of reduction with respect to contexts is obtained by the fact that graph morphisms allow the embedding of a graph within a larger one. Second, no distinct instance of the rules is needed, since graph isomorphism takes care of the closure with respect to structural congruence, and interfaces of the renaming of free names.

[3] Actually, five: since we consider mono matches, we need to assume an instance for the rules p_{in} and p_{out}, where the nodes labelled n and m may actually be coalesced

Our encoding is sound and complete with respect to the reduction relation $\rightarrow$, as stated by the theorem below.

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 subgraph with the *go* operator on the top. This allows the occurrence of reductions inside activated ambients, but not inside capabilities.

7 The Synthesized Transition System

In this section we apply the BC synthesis mechanism to $\mathcal{R}_{amb}$ in order to obtain a LTS for graphs representing MAs processes. We first show some examples of rewriting steps with BCs, then we use some pruning techniques (proposed in [4]) in order to obtain a simpler presentation of the derived LTS. This presentation is then used in the next section in order to define a LTS directly over MAs.

7.1 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 $P = (\nu n)(n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}])$. 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, we proceed by showing the transitions generated by such pairs.

First of all, take as D the left-hand side L_{in} and note that there is only one map into the graph G. The transition generated by this choice is depicted in Figure 8. The graph G^+ is the same as G. Now C and H are constructed as in a standard DPO rewriting step. When taking D as the whole left-hand side, $J \rightarrowtail G$ needs no context for the reaction and thus the label of this transition is the identity context, i.e., two isomorphisms into the discrete graphs with three nodes $\{p, a, m\}$.[4] Intuitively, this corresponds to an internal transition over processes, labelled with τ.

Now we take as D the subgraph of L_{in} representing an ambient with a capability *in* inside it. Note that also in this case there is only one possible map into the graph G. The resulting transition is shown in Figure 9. The graph G^+ is the graph G in parallel with the graph representing an ambient m, thus intuitively it represents the process $(\nu n)(n[in\ m.\mathsf{0}]|m[out\ m.\mathsf{0}]|m[X])$ for some process variable X. The graph $J \rightarrowtail G$, in order to reach the graph G^+, has to borrow from the environment the context $J \rightarrowtail F \leftarrowtail K$ that represents the syntactic context $-|m[X]$. Note that in the resulting interface K there is a process node $\bullet^{4_p}$ pointing to the process node of F occurring inside the ambient m. This process node in K represents the

[4] Or, equivalently, to the value of the expression $id_p \otimes id_a \otimes id_m$, as defined in Appendix B.

process variable X, as further detailed in Appendix E. The graphs C and H are then constructed as in the standard DPO approach. Intuitively, $K \to H$ represents the process $m[out\ m.0]|m[n[0]|X]$, where X is the same process variable occurring in the label $J \rightarrowtail F \leftarrowtail K$. This can be understood by observing that the process node $\bullet^{4_P}$ of K points both to a node of H and to a node of F. Summarizing, this transition moves the ambient n into an ambient m that is provided by the environment.

Another possible D is the subgraph of L_{in} consisting of the ambient depicted in the lower part of L_{in}. In this case, there are two possible maps into the graph G: the map into the subgraph of G representing the ambient m, and the map into the subgraph of G representing the restricted ambient n.

In the first case, we obtain the transition shown in Figure 10. The graph G^+ is the graph G in parallel with the graph representing a fresh ambient name w having inside a capability $in\ m$. Intuitively, it represents the extended process $(\nu n)(n[in\ m.0]|m[out\ m.0]|w[in\ m.X_2|X_1])$ for some process variables X_1, X_2. In order to reach G^+, the graph $J \rightarrowtail G$ has to borrow from the environment the context $J \rightarrowtail F \leftarrowtail K$ representing the syntactic context $-|w[in\ m.X_2|X_1]$. As in the above case X_1 and X_2 are process variables, since in the interface K there are the process nodes $\bullet^{2_P}$ and $\bullet^{3_P}$. The graphs C and H are obtained by a standard DPO derivation. The graph $K \to H$ represents the extended process $(\nu n)(n[in\ m.0]|m[out\ m.0|w[X_2|X_1]])$. Summarizing, this transition represents an ambient w from the environment entering inside the ambient m of the process P.

In the second case no transition is possible. Indeed the graph G^+ is the whole graph G in parallel with a fresh ambient w having inside a capability $in\ n$, but the pushout complement of $J \rightarrowtail G \rightarrowtail G^+$ does not exist, because n is restricted and thus it does not belong to the interface J. Intuitively, this means that no ambient from the environment can enter inside a restricted sibling ambient n.

In order to perform a complete analysis, we should consider all the pairs of monos $G \leftarrowtail D \rightarrowtail L_{in}$: we can avoid to check the others pairs not considered above by exploiting the pruning techniques presented in the next subsection.

7.2 Reducing the borrowing

In order to know all the possible transitions originating from a graph with interfaces $J \rightarrowtail G$, all the subgraphs D's of L_{in}, L_{out} and L_{open} should be analyzed. To shorten this long and tedious procedure, we use the two pruning techniques presented in [4].

The first one is based on the observation that those items of a left-hand side L that are not in D have to be glued to G through J. Let us consider a node n of D corresponding to a node n' in L, such that n' is the source or the target of some edge e that does not occur in D. Since the edge e is in L but not in D, it must be added to G through J, and thus n, called *boundary node*, must be also in J.

The notion of boundary nodes is formally captured by the categorical notion of *initial pushout* (defined in the Appendix C). Since our category has initial pushouts, the previous discussion is formalized by the lemma below.

Lemma 7.1 ([4]) *A graph with interfaces $J \to G$ can perform a BC rewriting step*

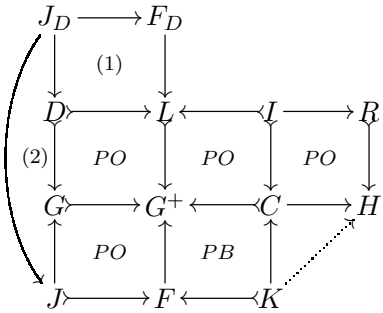


Fig. 4. The BC construction together with commuting squares (1) (the initial pushout of $D \rightarrowtail L$) and (2).

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 $D \rightarrowtail G$,*
- *a morphism $J_D \to J$ (where J_D is the initial pushout of $D \rightarrowtail L$) such that square (2) in Figure 4 commutes.*

This corollary allows to heavily prune the space of possible D's. As for graphs corresponding to the encoding of processes, we can exclude all those D's having a continuation process node (any node depicted by • that is not the root) as boundary node, observing that the only process node in the interface J is the root node.

A further pruning —partially based on proof techniques presented in [10]— is performed by excluding all those D's which generate a BC transition that is not relevant for the bisimilarity. In general terms, we may exclude all the D's that contain only nodes, since those D's can be embedded in every graph (with the same interface) generating the same transitions. Moreover, concerning our case study, those transitions generated by a D having the root node without the edge labelled *go* are also not relevant. In fact, 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.

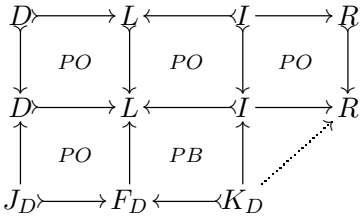
The two pruning techniques presented above allow us to only consider the partial matches D shown in Figures 5, E.1 and E.2.

7.3 Minimal transitions

In Section 7.2 we restricted the space of possible D's to those in Figures 5, E.1 and E.2. However reasoning on the synthesized LTS is still hard (this is usually the case when working with derived LTSs, as pointed out in [2] and [3], where the authors state that an SOS presentation of the synthesized LTS would be 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 7.2 and J_D containing only the boundary nodes of D.

The minimal transitions have the following shape



where the leftmost square in the lower row is an initial pushout.

Figures 5, E.1 and E.2 concisely represent these transitions, showing for each of these the starting graph D, the label $J_D \rightarrowtail F_D \leftarrowtail K_D$, and the resulting graph R. All the transitions originated from a graph $J \rightarrowtail G$ (representing a process) can be characterized by exploiting 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 Figure 4 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 $J_D \to J$) and all the name nodes $\circ$ representing the free names of the modeled process (as expected by our encoding). Then the graph F only contains the whole graph F_D and all the nodes of J (indeed, as shown in Proposition 2.5 of [4], F can be obtained as the pushout of $J_D \to F_D$ and $J_D \to J$). Moreover, it is easy to prove that K is a discrete graph containing all and only the nodes of F, or more concretely, K consists of the nodes of J and K_D.

Finally, the resulting graph H is obtained by replacing in the graph G the subgraph D with R (as shown in Proposition 2.5 of [4], it can be computed in a DPO step of $D \leftarrowtail D \cap I \to R$, where $D \cap I$ is the pullbacks of $D \rightarrowtail L$ and $I \rightarrowtail L$).

As an example, consider the BC rewriting step shown in Figure 8. It is derivable by the minimal transition for D_{in_4} (shown in Figure E.1). First of all note that there exist $D_{in_4} \to G$ and $\emptyset \to J$ such that the square (2) in Figure 4 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}.

8 A New LTS for Mobile Ambients

This section presents a LTS directly defined over MAs processes. The inference rules describing this LTS are obtained from the transitions of the LTS on graphs presented in Section 7.3. 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 presented in Figures 6 and 7.

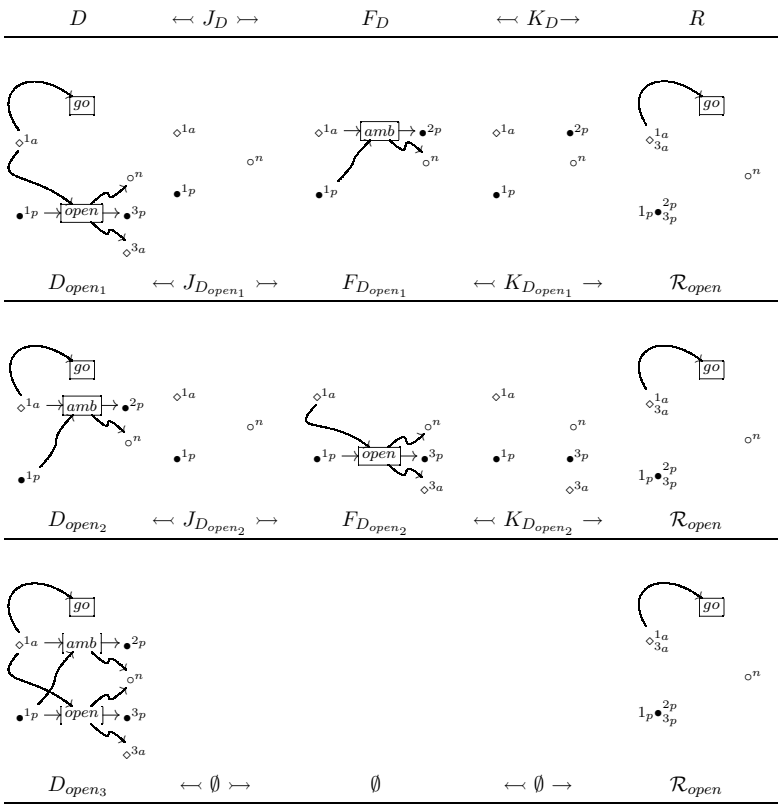


Fig. 5. The minimal transitions generated by the rule p_{open}.

8.1 *The labelled rules on processes...*

The rules in Figure 6 represent the τ-actions modeling internal computations. Note that the labels of the transitions are contexts composed of just a hole $-$, while the resulting states are processes over MAs standard syntax. 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 same transition relation of the reduction relation over MAs, thus they could be replaced with the rules in Table 3.

The rules in Figure 7 model the interactions of a process with its environment. Note that both labels and resulting states contain process and name variables. We define a LTS for processes over the standard syntax of mobile ambients by instantiating all the variables of the labels and of the resulting states. Formally, we say that $P \xrightarrow{l} Q$ (for l and Q pure processes) if and only if $P \xrightarrow{l_\epsilon} Q_\epsilon$ and there exists a substitution σ such that $Q_\epsilon\sigma \equiv Q$ and $l_\epsilon\sigma \equiv l$.

The rule Open models the opening of an ambient provided by the environment. In particular, 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 the ambients are provided by the environment. In the rule CoIn an ambient provided by the environment enters an ambient of the process. The rule OutAmb models an ambient of the process exiting from an ambient provided by the environment, while in the rule Out both ambients are provided by the environment.

Our LTS does not conform to the so-called SOS style: indeed, the premises of the inference rules are just constraints over the structure of the process. This depends on fact that the rules of our LTS are obtained from the borrowed minimal transitions. Each rule corresponds to one minimal transition presented in Section 7.3 and it is obtained as described below.

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

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

Fig. 6. The internal transitions (for $\mathcal{C}[-]$ context containing only ambients and parallel operators).

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

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

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

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

Fig. 7. The environmental transitions.

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 exist a mono $D \rightarrowtail G$, for some D of a minimal transition, and a morphism $J_D \to J$, such that square (2) in Figure 4 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, then also 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 Figure 5.

Consider a graph $J \rightarrowtail G$ representing the encoding for a process P. If there exist a mono $D_{open_1} \rightarrowtail G$ and a morphism $J_{D_{open_1}} \rightarrow J$, such that the square (2) in Figure 4 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 $J_{D_{open_1}} \rightarrow J$ (such that the square (2) in Figure 4 commutes) may exist if and only if $P \equiv (\nu A)(open\ n.P_1|P_2)$. Moreover, 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 represents 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. In order to have a deeper intuition about the correspondence between process variables and graphs, the interested reader is referred to Appendix E. Instead, Appendix D shows the derivation of the rule OUT.

Fig. 8. Ambient n enters ambient m. This corresponds to the transition $(\nu n)(n[in\ m.0]|m[out\ m.0]) \xrightarrow{-} (\nu n)(m[n[0]|out\ m.0])$.

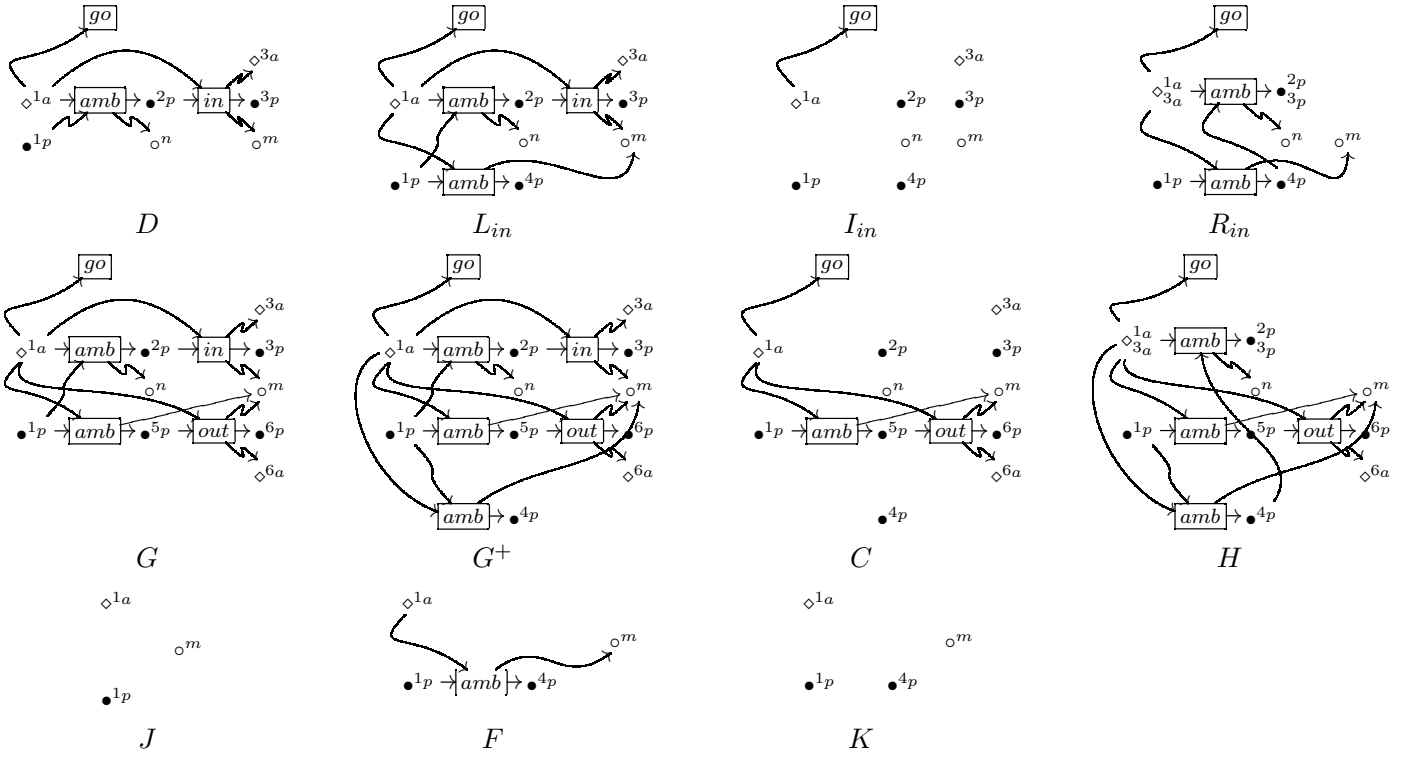


Fig. 9. Ambient n enters ambient m (from environment). This corresponds to the transition $(\nu n)(n[in\ m.0]|m[out\ m.0]) \xrightarrow{-|m[X]} (\nu n)(m[out\ m.0]|m[n[0]|X])$.

Fig. 10. Ambient w (from environment) enters ambient m. This corresponds to the transition $(\nu n)(n[in\ m.0]|m[out\ m.0]) \xrightarrow{-|w[in\ m.X_2|X_1]} (\nu n)(n[in\ m.0]|m[out\ m.0|w[X_2|X_1]])$.

9 Conclusions, related and future work

In this paper we exploit the graphical encoding for MAs, proposed in [12], to distill a LTS on (processes encoded as) graphs. We then use this LTS in order to infer a LTS directly defined on the processes of the MAs calculus. For the sake of simplicity, we considered a graphical encoding for MAs without communication primitives, as well as without recursive expressions. A graphical encoding for the whole calculus could be obtained by tackling both communication primitives and recursive processes along the lines of the solution in [4]. Once the graphical encoding for the whole calculus has been defined, the technique presented in this paper could be applied in order to obtain a LTS for the whole MAs calculus.

In spite of the great interest received by MAs, there are relatively few works concerning a labelled characterization of the calculus. After early attempts by Cardelli and Gordon [14] and (*via* a graphical encoding) Ferrari, Montanari and Tuosto [11], the only papers addressing this issue that we are aware of are [16] by Merro and Zappa-Nardelli, [23] by Rathke and Sobociński. The LTS of the former work is restricted to *systems*, i.e., those processes obtained by the parallel composition of ambients. For this reason, our rules In, Open and Out have not a counterpart in [16]. Instead, the rules InAmb, CoIn and OutAmb exactly correspond to the rules (Enter), (Co-Enter), (Exit) in Table 6 of [16]. 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 $k[-|\mathit{open}\ n.X_1|X_2]$ (again, this difference is due to the fact that the LTS of [16] is restricted to systems).

It is important to note that, differently from our LTS, the labels of the rules (Enter) and (Exit) contain the name of the migrating ambient n. This requires defining two extra rules (Enter Shh) and (Exit Shh) for the case when n is restricted.

Analogously to our work, Rathke and Sobociński employ a general systematic procedure for deriving LTSs that they have previously introduced in [20]. The detailed comparison is left as future work, but we conjecture that the two LTSs exactly correspond. Indeed, the seven axioms in Figure 6 of [23] are in one to one correspondence with our seven rules in Figure 7. The main difference concerns the derivation procedures that have been employed and the presentations of the resulting LTSs. Theirs is presented in a SOS style (as a result of their procedure), while ours relies on the structural congruence (as a result of the BC mechanism applied to the graphical encoding). Their style carries more information than ours, since it describes the behaviour of each syntactic operator, but our presentation seems more intuitive, since it employs fewer compact rules (10 instead of their 27).

Beside the presentation of a succinct LTS for mobile ambients, our work is a relevant case study for the theory of reactive systems [15]. As already pointed out in the introduction, BC rewriting and bigraphical reactive systems [19] are both instances of this theory. This paper, together with [4], shows that the BCs approach is quite effective in deriving LTS for process calculi.

In particular, this work confirms the advantage of BCs over graphs with interfaces with respect to bigraphs. In bigraphs, all the reduction rules must be ground

(i.e., they can not contain process variables). As a result, also the labels and the arriving states of the derived transitions must be ground. Instead, rewriting with BCs allows to employ few non ground rules (as shown in this paper) and thus 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, because the variables in the labels always occur "outside" of the arriving state and thus can be forgotten. As an example, consider the CCS transition $a.b \xrightarrow{-|\bar{a}.Y} b|Y$ derived from the (non ground) reduction rule $a.X|\bar{a}.Y \rightarrow X|Y$. The behaviour of the process $b|Y$ is trivially equivalent to b: their interaction is basically restricted to processes offering a $\bar{b}$ action, and we can thus avoid to consider Y. Instead, in the case of mobile ambients, the ability of considering 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 [3] 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 and not just the minimal ones) is often more adequate. This is the case of Logic Programming, open π-calculus [5] and Petri nets [6]. The present work provides a further successful test of the above claim. Indeed, it is easy to see that (the standard notion of) bisimilarity over our LTS is too strict, because it allows to observe the ability of an ambient to migrate, while it should be unobservable, as pointed out in [16]. 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 us the *semi-saturated bisimulation* [5]. Instead of requiring that two bisimilar processes must perform transitions with the same label, the definition of 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 second and third points of Definition 3.2 in [16] has exactly this shape (the labels $*.enter_n$ and $*.exit_n$ correspond to our contexts $-|n[X_1]$ and $n[-|X_1]$). We leave as future work to exploit this intuition and to check if (semi-)saturated bisimulation on our LTS corresponds to the behavioral equivalence proposed by Merro and Zappa Nardelli.

References

[1] Baldan, P., 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*, Handbook of Graph Grammars and Computing by Graph Transformation **3**, World Scientific, 1999 pp. 107–187.

[2] Baldan, P., 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*, Lect. Notes in Comp. Sci. **4178** (2006), pp. 153–167.

[3] Bonchi, F., "Abstract Semantics by Observable Contexts," Ph.D. thesis, Department of Informatics, University of Pisa (2008).

[4] Bonchi, F., 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*, Lect. Notes in Comp. Sci. **4178** (2006), pp. 168–183.

[5] Bonchi, F., B. König and U. Montanari, *Saturated semantics for reactive systems*, in: *Logic in Computer Science* (2006), pp. 69–80.

[6] Bonchi, F. and U. Montanari, *Coalgebraic models for reactive systems*, in: L. Caires and V. Vasconcelos, editors, *Concurrency Theory*, Lect. Notes in Comp. Sci. **4703** (2007), pp. 364–379.

[7] Cardelli, L. and A. Gordon, *Mobile ambients*, Theor. Comp. Sci. **240** (2000), pp. 177–213.

[8] Corradini, A. and F. Gadducci, *An algebraic presentation of term graphs, via gs-monoidal categories*, Applied Categorical Structures **7** (1999), pp. 299–331.

[9] Corradini, A., U. Montanari and F. Rossi, *Graph processes*, Fundamenta Informaticae **26** (1996), pp. 241–265.

[10] Ehrig, H. and B. König, *Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts*, Mathematical Structures in Computer Science **16** (2006), pp. 1133–1163.

[11] Ferrari, G., 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*, Lect. Notes in Comp. Sci. **2202** (2001), pp. 1–16.

[12] Gadducci, F. and G. V. Monreale, *A decentralized implementation of mobile ambients*, in: R. Heckel and G. Taentzer, editors, *Graph Transformation*, Lect. Notes in Comp. Sci. **forthcoming** (2008).

[13] Gadducci, F. 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. de Vrijer, editors, *Processes, terms and cycles (Klop Festschrift)*, Lect. Notes in Comp. Sci. **3838** (2005), pp. 106–126.

[14] Gordon, A. D. and L. Cardelli, *Equational properties of mobile ambients*, Mathematical Structures in Computer Science **13** (2003), pp. 371–408.

[15] Leifer, J. and R. Milner, *Deriving bisimulation congruences for reactive systems*, in: C. Palamidessi, editor, *Concurrency Theory*, Lect. Notes in Comp. Sci. **1877** (2000), pp. 243–258.

[16] Merro, M. and F. Zappa Nardelli, *Behavioral theory for mobile ambients*, Journal of the ACM **52** (2005), pp. 961–1023.

[17] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

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

[19] Milner, R., *Pure bigraphs: Structure and dynamics*, Information and Computation **204** (2006), pp. 60–122.

[20] Rathke, J. and P. Sobociński, *Deconstructing behavioural theories of mobility*, in: G. Ausiello, J. Karhumäki, G. Mauri and L. Ong, editors, *Theoretical Computer Science*, Lect. Notes in Comp. Sci. **forthcoming** (2008).

[21] Regev, A., E. Panina, W. Silverman, L. Cardelli and E. Shapiro, *Bioambients: an abstraction for biological compartments*, Theor. Comp. Sci. **325** (2004), pp. 141–167.

[22] Sobociński, P., "Deriving bisimulation congruences from reduction systems," Ph.D. thesis, BRICS, Department of Computer Science, University of Aarhus (2004).

[23] Sobociński, P. and J. Rathke, *Deriving structural labelled transitions for mobile ambients*, in: F. van Breugel and M. Chechik, editors, *Concurrency Theory*, Lect. Notes in Comp. Sci. **5201** (2008), pp. 462–476.

A Two binary operators

Definition A.1 (two composition 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. Then, 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 on nodes 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* $\mathbb{H} = J' \xrightarrow{j'} H \xleftarrow{k'} K'$ *be graphs with discrete, compatible interfaces.*[5] *Then, their* parallel composition *is the graph with discrete interfaces* $\mathbb{G} \otimes \mathbb{H} = (J \cup J') \xrightarrow{j''} V \xleftarrow{k''} (K \cup K')$*, for* V *the disjoint union* $G \uplus H$*, modulo the equivalence on nodes 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 $\mathbb{G} \circ \mathbb{G}'$ is obtained by taking the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{G}'$, and gluing the outputs of $\mathbb{G}$ with the corresponding inputs of $\mathbb{G}'$. Similarly, the parallel composition $\mathbb{G} \otimes \mathbb{H}$ is obtained by taking the disjoint union of the graphs underlying $\mathbb{G}$ and $\mathbb{H}$, and gluing the inputs (outputs) of $\mathbb{G}$ with the corresponding inputs (outputs) of $\mathbb{H}$. The operations are defined on "concrete" graphs, even if the result is independent of the choice of representatives.

B Process encoding

Figures B.1 and B.2 depict a class of graphs such that all processes can be encoded into an expression containing only those graphs as constants, and parallel and sequential composition as binary operators. We assume $p, a \notin \mathcal{N}$ and $n \in \mathcal{N}$.

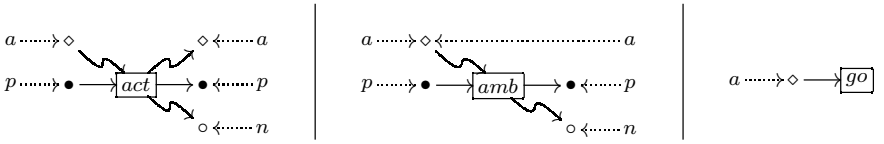


Fig. B.1. Graphs act_n (with $act \in \{in, out, open\}$); amb_n; and go (left to right).

In the following, 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. 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. Note that the last expression is well defined, because the $\otimes$ operator is associative. The definition below introduces the encoding of processes into graphs with interfaces, mapping a process into a graph expression. Note that the encoding $[\![M.P]\!]_\Gamma$ represents the encoding of *in* $n.P$, *out* $n.P$ and *open* $n.P$, while act_n represents the in_n, out_n and $open_n$ graphs, respectively.

Definition B.1 (Encoding for processes) *Let* P *be a pure process and let* Γ *be a set of names such that* $fn(P) \subseteq \Gamma$*. The* encoding *of* P*, denoted by* $[\![P]\!]_\Gamma$*, is defined by structural induction according to the following rules*

$$
\begin{array}{lcllcl}
[\![\mathtt{0}]\!]_\Gamma & = & \mathtt{0}_{a,p} \otimes \mathtt{0}_\Gamma & [\![n[P]]\!]_\Gamma & = & (amb_n \otimes id_\Gamma) \circ [\![P]\!]_\Gamma \\
[\![M.P]\!]_\Gamma & = & (act_n \otimes id_\Gamma) \circ [\![P]\!]_\Gamma & [\![P|Q]\!]_\Gamma & = & [\![P]\!]_\Gamma \otimes [\![Q]\!]_\Gamma \\
[\![(\nu n)P]\!]_\Gamma & = & (new_m \otimes id_\Gamma \otimes id_{a,p}) \circ [\![P\{^m/_n\}]\!]_{\Gamma\cup\{m\}} & & & \text{for } m \notin \Gamma
\end{array}
$$

Given a pure process P and a set of names Γ such that $fn(P) \subseteq \Gamma$, its *enriched encoding* is the graph $[\![P]\!]_\Gamma \otimes go$. We denote it by $[\![P]\!]^{go}_\Gamma$.

C Initial Pushout

Here we briefly report the definition of initial pushout. Note that the category of (typed hyper-)graphs we work in has initial pushouts for all arrows.

Definition C.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*

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

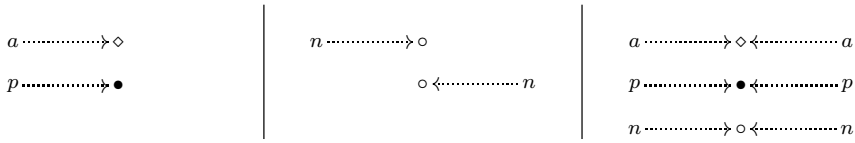
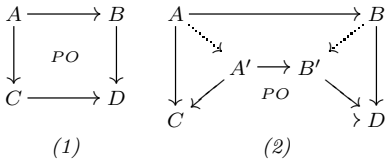


Fig. B.2. Graphs $\mathtt{0}_a$ and $\mathtt{0}_p$; $\mathtt{0}_n$ and new_n; id_a, id_p and id_n (top to bottom and left to right).

$B \to B'$ *such that diagram (2) commutes.*



D Minimal transitions

In Section 8 we explained how the rules of Figures 6 and 7 are derived from the minimal transitions of Figures 5, E.1 and E.2. Roughly, each rules corresponds to a minimal transition for a certain D. The premise of the rule for a process P corresponds to the existence of the monos $D \rightarrowtail G$ and $J_D \rightarrowtail J$ such that square (2) in Figure 4 commutes (for $J \rightarrowtail G = [\![P]\!]$). The conclusion of the rule states that P can perform a transition with a certain context, that is the label $J_D \rightarrowtail F_D \leftarrowtail K_D$ of the minimal transition, and then arrives in a process (over the extended syntax) that is the arriving state $K_D \to R$ of the minimal transition.

The reader should notice that while there are 13 minimal transitions, only 10 rules occur in Figures 6,7. 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}, while Out is generated by D_{out_1} and D'_{out_1}. We show the latter, since the others are analogous.

In the minimal transition D_{out_1} two ambients are borrowed from the environment. The first one has name m (i.e., the name from which the process want to exit), while the second has a fresh name n. This transition corresponds to the rule

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

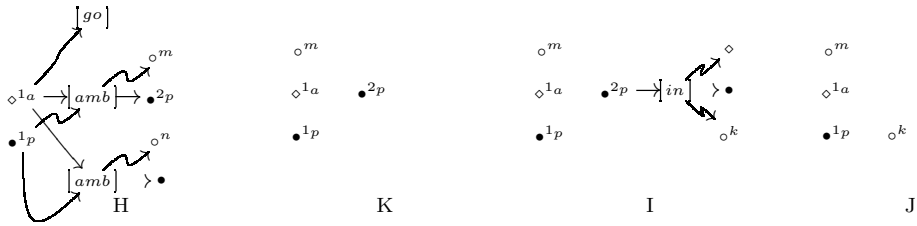
In the minimal transition D'_{out_1} the name n belongs to the process (it occurs inside the graph $D_{out'_1}$) but, since the node n occur in $J_{D'_{out_1}}$, it should appear in the interface J, i.e., it must be free. Thus, this transition corresponds to the rule

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

Now, notice that the conclusions of the two rules are identical. Thus we can put together the premises of the two rules above, and we get that n is a name variable. This is exactly the rule Out of Figure 7.

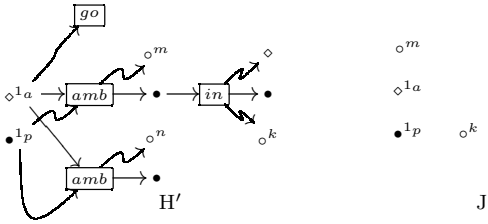
E Process variables, graphically

In the proposed LTS we heavily relied on process variables. These are derived directly from the arriving states of the minimal transitions. Here we explain the intuition underlying the use of process variables. Consider the graph $K \rightarrowtail H$ depicted below. This represents the (extended) process $(\nu n)(m[X_1]|n[0])$.



Indeed, the node $\bullet^{2_p}$ is in the interface (it corresponds to the process variable X_1) and this allows to instantiate X_1. As an example, the substitution $\{^{in\ k.0}/_{X_1}\}$ can be represented by the graph with interfaces $K \rightarrowtail I \leftarrowtail J$ depicted above. The process obtained by applying such substitution to $(\nu n)(m[X_1]|n[0])$ is $(\nu n)(m[in\ k.0]|n[0])$, corresponding to the graph $H' \leftarrowtail J$ (depicted below) that is obtained by composing

the graph $H \leftarrowtail K$ with $K \rightarrowtail I \leftarrowtail J$ (i.e., by pushing out $H \leftarrowtail K \rightarrowtail I$).



Analogously to the substitution of process variables, our composition does do not capture bound names. Consider e.g. the bound name n of $H \leftarrowtail K$. It does not appear in the interface K and thus, for all graph with interfaces $K \leftarrowtail I' \rightarrowtail J'$ (representing possible substitutions), it can not be identified with any name of I'.

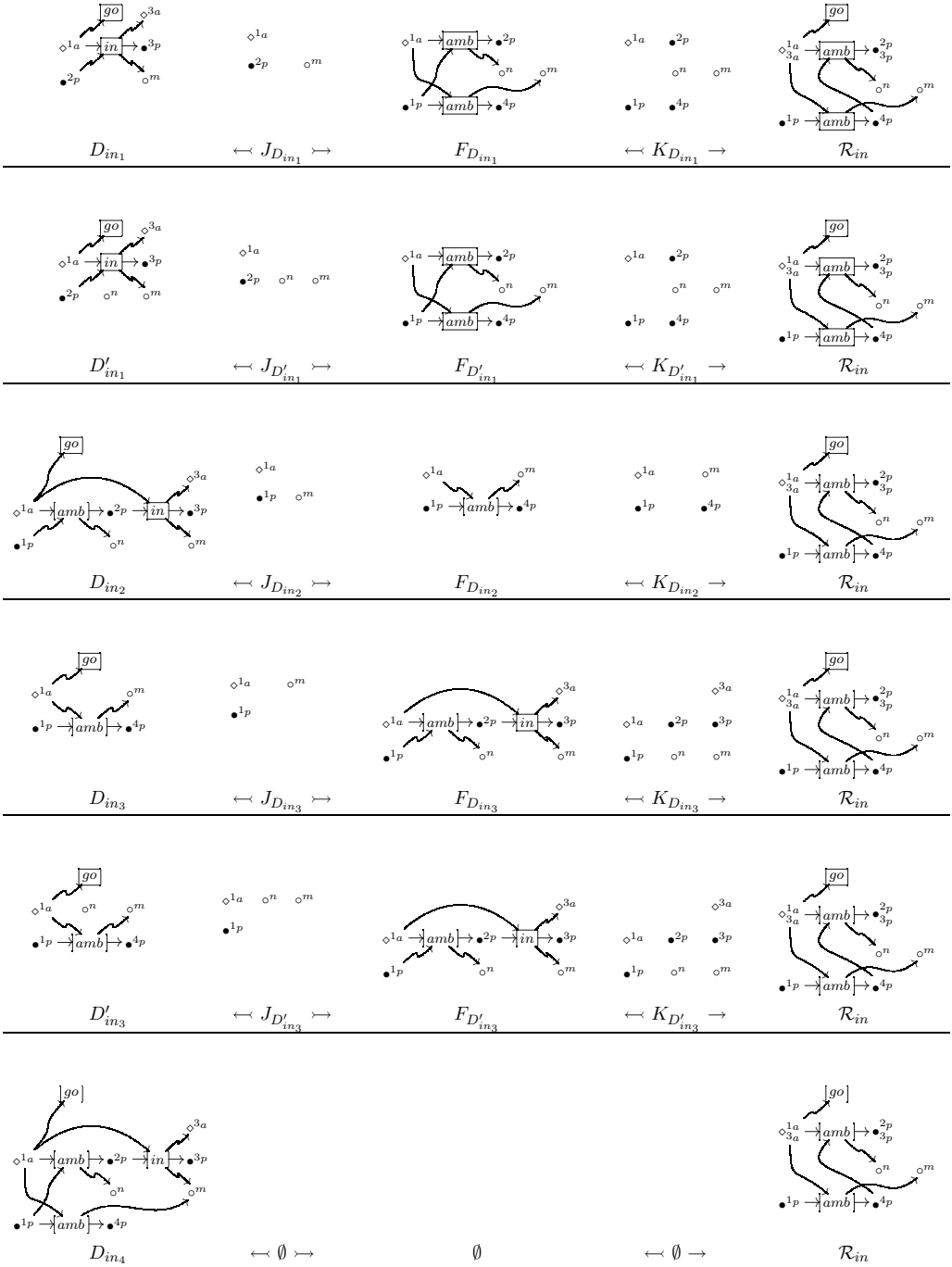


Fig. E.1. The minimal transitions generated by the rule p_{in}.

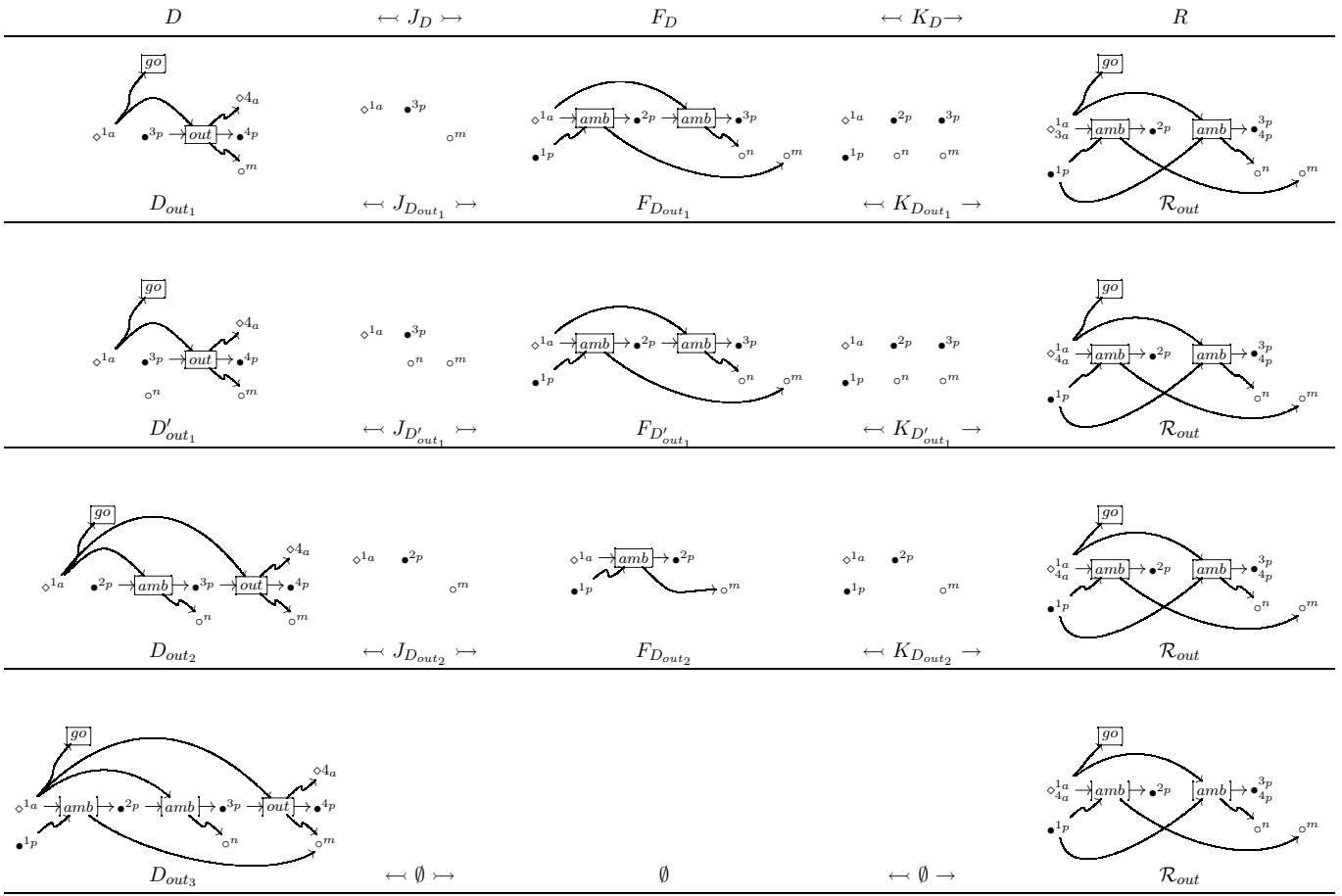


Fig. E.2. The minimal transitions generated by the rule p_{out}.