

A LTS Semantics of Ambients via Graph Synchronization with Mobility^{*}

GianLuigi Ferrari, Ugo Montanari, and Emilio Tuosto

Dipartimento di Informatica, Università di Pisa
{giangi,ugo,etuosto}@di.unipi.it

Abstract. We present a simple labelled transition system semantics of Cardelli and Gordon's Ambient calculus. We exploit a general and flexible model based on (hyper)graphs, where graph transformation is obtained via (hyper)edge replacement and local synchronization with mobility. In addition to tree-like ambients, the calculus we define works just as well with graph-like ambients, which are a more realistic model of internetworks.

1 Introduction

Foundational research on global computing aims at describing, modeling and analyzing the complex interactions taking place in internetwork applications encompassing several physical networks, multiple administration domains and a variety of possible users. Several models have been proposed to tackle the new computational phenomena. They usually take the form of distributed process calculi (e.g. Join calculus [7], Ambient calculus [2]), of specialized program logics (e.g. Mobile Unity [13], Mob_{adtl} [6]), and of Linda-like coordination languages (e.g. KLAIM [4], Lime [12]), to mention a few.

Most models mainly focus on the *spatial structure* of global computing. To reflect the idea of administration domains, they exhibit explicit localities, which help modeling distributed computations and the discovery of network resources and services. These features distinguish the models of global computing from the traditional models (and paradigms) for distributed programming (e.g. CORBA), the motto being *network awareness*: localities are under programmer's control.

However network awareness is only one relevant tile of the mosaic of global computing. Another important aspect concerns the *temporal structure* of the applications. The run-time environment typically interleaves computational activities with structuring and managing activities. The temporal structure of applications takes care of describing application rearrangements and security checks. A proper understanding of both spatial and temporal structures is clearly needed to allow formal verification of applications.

^{*} Partially supported by CNR project *Metodi per Sistemi Connessi mediante Reti*; by MURST project *Theory of Concurrency, Higher Order and Types*; by TMR Network GETGRATS; and by Esprit Working Groups *APPLIGRAPH*.

The Ambient calculus is one of the best studied models addressing the needs of global computing, and it has acquired the role of touchstone for the most recent proposals. However the interactive, abstract semantics of ambients is still not fully explored. In fact, as it is the case of most foundational calculi for global computing, reduction semantics for ambients has been found to be simpler than the corresponding labeled transition system (LTS) semantics. However, reduction semantics has the main disadvantage with respect to LTS semantics that it makes harder to define, and reason about, abstract compositional behavior.

A LTS operational semantics for ambients has been defined by Gordon and Cardelli in an unpublished note [1]. It requires the introduction in the calculus of co-actions, abstractions, concretions and outcomes. At the authors' knowledge, the bisimilarity abstract semantics based on this operational semantics has not been compared with the reduction semantics and with the logics developed by Cardelli, Gordon and Caires, which is equipped with specialized modalities to deal with the spatial and the temporal dimensions of global computing. Sewell [14] introduces a technique to develop an LTS-based semantics from a reduction semantics; however the resulting transition semantics exploits arbitrary contexts and, moreover, it is not inductive on process operators.

In this paper we define a LTS semantics of ambients by exploiting a general and flexible model based on (hyper)graphs, where graph transformation is obtained via (hyper)edge replacement and local synchronization with mobility. While getting acquainted with the formal techniques necessary for handling graphs (rather than trees or terms) may require some effort, the actual definition of the Ambient calculus is quite short and intuitive. Moreover, in addition to tree-like ambients, the calculus we define works just as well with graph-like ambients, which are a more realistic model of internetworks.

More generally, we propose our graph-based technique as a tool for modeling internetworking systems. In fact, edges can be used to represent components and nodes to model the network environment of components. Some edges sharing a node means that the corresponding components may interact by exploiting network communication infrastructure. Structured versions of graphs (typed graphs, term graphs, hierarchical graphs) can precisely model complex internetwork configurations and access policies.

Graph synchronization adds to network awareness the ability of dealing with the temporal dimension of computations. Graphs synchronization is purely local and it is obtained by the combination of graph rewriting with constraint solving. The intuitive idea is that local rewritings depends on the outcome of a (possibly global) constraint satisfaction algorithm. Mobility allows to exchange nodes during synchronizations, and thus constraint solving must include unification to allow for node binding.

One may wonder if this approach is too abstract and general and it does not capture the intrinsic limitations of internetworking applications. We feel that on the one side the generality of the approach can be tamed and adapted to the needs of the various layers of applications, more powerful primitives being made available to upper layers, like B2B or CSCW. On the other side, some impor-

tant network technologies actually require the solution of global constraints, like modifying local router tables according to the routing update information sent by the adjacent routers.

Graph rewriting based on edge replacement and synchronization was introduced in [3,5] and related to distributed constraint satisfaction problems in [11]. The version with mobility, which employs a notation based on logical sequents and inference rules, was introduced recently in [8] and extended in [9] to encode π -calculus. Abstract semantics based on bisimilarity was discussed in [10]. To model Ambient calculus, synchronized hyperedge replacement has been further extended in this paper with fusions. Fusions allow to coalesce in the right member of a production sets of interface nodes which are distinct in the left member. This extension is necessary for representing the effect of the *open* capability, which merges the localities inside and outside the open ambient.

In the paper we handle a limited version of ambients, without restriction (of ambient names) and process communication, and with guarded recursion rather than replication. We relate the operational semantics of ambients based on synchronized edge replacement to the original reduction semantics. We show that there is a bijective correspondence between ambient processes and certain graphs called ambient graphs. We also show that ambient processes and their corresponding graphs have corresponding reductions. Of course the graphs have in addition transitions with observable labels, which can be exploited in the abstract semantics, that however is not studied in the paper.

2 Hypergraphs and Graph Synchronization

We first review (as presented in [9]) the notion of hypergraph and its formalization in terms of well formed syntactic judgements. Then we introduce the notion of graph synchronization.

A *edge*, or simply an edge, is an atomic item with a label (from a ranked alphabet $LE = \{LE_n\}_{n=0,1,\dots}$) and with as many (ordered) tentacles as the rank of its label. A set of *nodes* together with a set of such edges forms a *hypergraph* (or simply a graph) if each edge is connected, by its tentacles, to its *attachment* nodes. A graph is equipped with a set of external nodes identified by distinct names. External nodes can be seen as the connecting points of a graph with its environment.

Now, we present a definition of graphs as *syntactic judgements*, where nodes correspond to names, external nodes to free names and edges to basic terms of the form $L(x_1, \dots, x_n)$, where x_i are arbitrary names and $L \in LE$.

Definition 1 (Graphs as Syntactic Judgements). *Let \mathcal{N} be a fixed infinite set of names and LE a ranked alphabet of labels. A syntactic judgement (or simply a judgement) is of the form $\Gamma \vdash G$ where,*

1. $\Gamma \subseteq \mathcal{N}$ is a set of names (the external nodes of the graph).
2. G is a term generated by the grammar
$$G ::= L(\mathbf{x}) \mid G|G \mid (\nu y)G \mid nil$$
where \mathbf{x} is a vector of names, L is an edge label with $rank(L) = |\mathbf{x}|$ and y is a name.

The correspondence theorem expressing that well-formed syntactic judgements up to structural axioms are isomorphic to graphs up to isomorphism has been proved in [9].

We now introduce the notion of synchronized edge replacement. Synchronized edge replacement is obtained using graph rewriting combined with constraint solving. More specifically, we use *context-free* productions enriched with actions that are used to coordinate the simultaneous application of various productions.

The following definitions introduce synchronized edge replacement systems where actions can declare and refer to names as nodes and where names are bound via unification.

A *context-free edge replacement production* rewrites a single edge into an arbitrary graph. A production $p = (L \rightarrow R)$ can be applied to a graph G yielding H if there is an occurrence of an edge labeled by L in G . Graph H is obtained from G by removing the previously matched edge and by embedding a fresh copy of R in G by coalescing its external nodes with the corresponding attachment nodes of the replaced edge. This notion of edge replacement yields the basic steps in the derivation process of an edge replacement grammar.

To model synchronized rewriting, it is necessary to add some labels to the nodes in productions. Assuming to have a ranked alphabet Act of actions, then we associate actions to some of the attachment nodes of the left member of the production. In this way, each rewrite of an edge must synchronize actions with (a number of) its adjacent edges and then all the participants will have to move as well (how many depends on the synchronization policy). It is clear that coordinated rewriting will allow the propagation of synchronization all over the graph where productions are applied. Determining which productions can be synchronized at any given stage corresponds to solve a distributed constraint satisfaction problem [11].

A *synchronized edge replacement grammar*, or simply a grammar, consists of an initial graph and a set of productions. A derivation is obtained by starting with the initial graph and by executing a sequence of transitions, each obtained by synchronizing possibly several productions.

Now, for adding mobility to our model of computation we let a production to declare on each of its connecting nodes new names for the nodes it creates and to share these names and/or other existing names with the rest of the graph using the synchronization process. This is done in a production by adding to the action in a node a tuple of names that one wants to communicate. Therefore, the synchronization of a rewriting rule has to match not only actions, but also has to unify the tuples of names. After the productions are applied, the declared names that were unified are used to obtain the final graph by merging the corresponding nodes.

The expressive power of our model depends on the meaning of the names unified in the synchronization process. If these names correspond only to nodes newly generated in the productions, the expressive power is analogous to the π - I -calculus, where only extruded names can be transmitted. Instead, if also “old” nodes can be communicated, but not unified, we are analogous to the

π -calculus. If all types of nodes can be unified, the corresponding process algebra is the fusion calculus [15]. However we emphasize that in our model it is possible (and easy) to define multiple synchronizations, while the existing calculi are usually limited to binary synchronizations.

In this paper we handle for the first time the general case (with Milner synchronization style). The $\pi - I$ -like case was defined in [8,10] while the intermediate case was presented in paper [9], which in fact includes the encoding of the π -calculus.

Below we define the transitions of a grammar as certain logical sequents. We exploit the previously introduced representation of graphs as syntactic judgements. Notice that no distinction is made between nodes and names.

Definition 3 (Transitions (with fusion)). *A transition has the form*

$$\Gamma \vdash G_1 \xrightarrow{\Lambda, \pi} \phi \vdash G_2$$

where:

1. $\Lambda : \Gamma \rightarrow (Act \times \mathcal{N}^*)$
2. $\pi : \Gamma \rightarrow \Gamma$ and $x \in \pi^{-1}(x)$
3. $n(\Lambda) = \{z \mid \exists x. \Lambda(x) = (a, \mathbf{y}), z \in Set(\mathbf{y})\}$
4. $\Delta = n(\Lambda) - \Gamma$
5. $\phi = \pi(\Gamma) \cup \Delta$

A transition says that G_1 is rewritten into G_2 satisfying a *set of requirements* Λ and a *fusion substitution* π . The free nodes of graph G_2 must include the free nodes of G_1 (after applying π) and those new nodes (Δ) that are used in synchronization. Note that ϕ is determined by the Γ and Λ of the same transition.

The set of requirements $\Lambda \subseteq \Gamma \times Act \times \mathcal{N}^*$ is defined as a partial function in its first argument, i.e. if $(x, a, \mathbf{y}) \in \Lambda$ we write $\Lambda(x) = (a, \mathbf{y})$ with $rank(a) = |\mathbf{y}|$. With $\Lambda(x) \uparrow$ we mean that the function is not defined for x , i.e. that there is no requirement in Λ with x as first argument. Function $set(\mathbf{y})$ returns the set of names in vector \mathbf{y} . The definition of Λ as a function means that all edges in G_1 attached to node x that are participating in a synchronization must satisfy the conditions of the corresponding synchronization algebra. The function is partial since not all nodes need to be loci of synchronization.

Fusion substitution π determines a partition of Γ where all nodes in an equivalence class are mapped to a representative element of the class. We use $x \mapsto y$ to denote the substitution mapping node x to y .

Definition 4 (Productions). *A synchronized production, or simply a production, is a special transition of the form,*

$$x_1, \dots, x_n \vdash L(x_1, \dots, x_n) \xrightarrow{\Lambda, \pi} \phi \vdash G$$

The context-free character of productions is here made clear by the fact that the graph to be rewritten consists of a single edge with distinct nodes. Productions combine the roles of prefix, sum and recursion in process calculi.

Renaming can be applied to productions in several ways: i) free names x_1, \dots, x_n can be changed throughout the sequent; ii) names declared in $\Delta = n(\lambda) - \Gamma$ can be α -converted; and iii) the representative names chosen by π can be consistently changed. Also *identity productions* of the form

$$x_1, \dots, x_n \vdash L(x_1, \dots, x_n) \xrightarrow{\emptyset, id} x_1, \dots, x_n \vdash L(x_1, \dots, x_n)$$

are always considered available.

Definition 5 (Grammars). Let \mathcal{N} be a fixed infinite set of names, LE a ranked alphabet of labels and Act a ranked set of actions. A grammar consists of an initial graph $\Gamma_0 \vdash G_0$ and a set \mathcal{P} of productions on LE and Act .

A derivation is a finite or infinite sequence of the form $\Gamma \vdash G \xrightarrow{\Lambda_1, \pi_1} \phi_1 \vdash G_1 \xrightarrow{\Lambda_2, \pi_2} \dots \xrightarrow{\Lambda_n, \pi_n} \phi_n \vdash G_n \dots$, where $\Gamma \vdash G \xrightarrow{\Lambda_1, \pi_1} \phi_1 \vdash G_1$ and $\phi_{i-1} \vdash G_{i-1} \xrightarrow{\Lambda_i, \pi_i} \Gamma_i \vdash G_i$, $i = 2, \dots, n$ are transitions in the set $T(\mathcal{P})$ of transitions generated by \mathcal{P} . Transitions $T(\mathcal{P})$ are generated by \mathcal{P} applying the inference rules defined below.

Definition 6 (Inference rules). Let $\langle \Gamma \vdash G_0, \mathcal{P} \rangle$ be a grammar. The set $T(\mathcal{P})$ of transitions is obtained from the productions P using the inference rules in Table 2 where the side conditions of the rules are:

$$\psi_1 \stackrel{\text{def}}{\iff} \left\{ \begin{array}{l} \Delta \cap \sigma(\Gamma) = \emptyset \text{ and } \forall x \in \Delta, \sigma(x) = x \\ \sigma(x) = \sigma(y) \wedge \Lambda(x) \downarrow \wedge \Lambda(y) \downarrow \wedge x \neq y \Rightarrow \\ \quad (\forall z \notin \{x, y\}, \sigma(z) = \sigma(x) \Rightarrow \Lambda(z) \uparrow) \\ \quad \wedge \Lambda(x) = (a, \mathbf{v}) \wedge \Lambda(y) = (\bar{a}, \mathbf{w}) \wedge a \neq \tau \\ \rho = \text{mgu}(\{\sigma(\mathbf{v}) = \sigma(\mathbf{w}) \mid \sigma(x) = \sigma(y) \wedge \Lambda(x) = (a, \mathbf{v}) \wedge \Lambda(y) = (\bar{a}, \mathbf{w})\} \\ \quad \cup \{\sigma(x) = \sigma(y) \mid \pi(x) = \pi(y)\}) \\ \Lambda'(z) = \begin{cases} (\tau, \langle \rangle), & \text{if } \sigma(x) = \sigma(y) = z \wedge x \neq y \wedge \Lambda(x) \downarrow \wedge \Lambda(y) \downarrow \\ \rho(\sigma(\Lambda))(z), & \text{otherwise} \end{cases} \\ \pi'(\sigma(x)) = \rho(\sigma(\pi(x))) \\ \mathbf{u} = \rho(\sigma(\phi)) - \phi' \end{array} \right.$$

$$\psi_2 \stackrel{\text{def}}{\iff} \left\{ \begin{array}{l} (\pi(x) = \pi(y) \wedge x \neq y) \Rightarrow \pi(x) \neq x \\ \Lambda(x) \uparrow \text{ or } \Lambda(x) = (\tau, \langle \rangle), \\ \Lambda' = \Lambda - (x, \tau, \langle \rangle) \\ \mathbf{z} = \phi - \phi' \end{array} \right.$$

Rule (*par*) simply combines together two disjoint judgements.

Rule (*merge*) is the rule for synchronization. The rule states that in a transition it is possible to merge two nodes x and y that offer complementary non-silent actions (conditions on σ). Here ρ is the most general unifier that fuse the corresponding names of the actions and propagates the previous fusions (determined by π). The label Λ' takes into account all possible synchronizations and leaves unchanged the actions offered on the other nodes up to the necessary fusions (ρ and σ). The new fusion substitution π' acts on $\sigma(\Gamma)$ by applying to it the mgu ρ . Finally, the names in ϕ after the fusion which are not present in

Table 2. Inference rules for graph synchronization

$$\begin{array}{c}
(\text{par}) \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda, \pi} \phi \vdash G_2 \quad \Gamma' \vdash G'_1 \xrightarrow{\Lambda', \pi'} \phi' \vdash G'_2}{\Gamma, \Gamma' \vdash G_1 | G'_1 \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \phi, \phi' \vdash G_2 | G'_2} \quad \text{where } \Gamma \cap \Gamma' = \emptyset \\
\\
(\text{merge}) \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda, \pi} \phi \vdash G_2}{\sigma \Gamma \vdash \sigma G_1 \xrightarrow{\Lambda', \pi'} \phi' \vdash \nu \mathbf{u}.\rho(\sigma(G_2))} \quad \text{where } \psi_1 \text{ holds} \\
\\
(\text{res}) \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda, \pi} \phi \vdash G_2}{\Gamma \vdash \nu x.G_1 \xrightarrow{\Lambda', \pi|_{\Gamma}} \phi' \vdash \nu \mathbf{z}.G_2} \quad \text{where } \psi_2 \text{ holds}
\end{array}$$

$\phi' = \pi'(G) \cup (n(\Lambda') - \sigma(G))$ are restricted; this corresponds to the close rule of the π -calculus.

Rule (*res*) deals with node restriction. According to the first condition, the restricted node must not be the representative element of its equivalence class induced by π when this class contains nodes different from x . Furthermore, only nodes can be restricted where either no action or only synchronization actions take place. If this conditions hold, Λ' is obtained by hiding the (possible) silent action on x and restricting all the nodes that are not in ϕ' . Notice that ϕ' is defined as usual as $\phi = \pi(G) \cup \Delta$, with $\Delta = n(\Lambda') - G$.

3 Ambient Calculus

In this section we apply our graph synchronization framework to the Ambient calculus [2] that is considered one of the most suitable calculi for representing wide area network computations. First we give syntax and semantics of the Ambient calculus and then its representation in terms of graphs is specified.

3.1 The Calculus

The syntax and the reduction semantics of Ambient calculus [2] is given below. The calculus relies on the notion of *ambient* that can be thought of as a bounded environment where processes interact. An ambient has a name, a collection of local agents and a collection of subambients. Ambients can be moved as a whole under the control of agents which are confined to ambients. Processes use *capabilities* for controlling interaction. We do not consider synchronization and restriction, and replication is replaced by (guarded) recursion.

Definition 7 (Syntax). *Let N be an infinite set of names ranged over by $a, b, c, \dots, n, m, p, r, \dots$; let X, Y, Z, \dots be process variables.*

$$\begin{array}{l}
M ::= \text{in } n \mid \text{out } n \mid \text{open } n \\
P, Q ::= 0 \mid n[P] \mid M.P \mid P|Q \mid \text{rec } X.P \mid X
\end{array}$$

We assume that X is guarded by M in $\text{rec } X.P$.

We denote with Proc the set of the Ambient calculus processes.

Capabilities M are the usual Ambient calculus capabilities: $\text{in } n$ allows to drive an ambient inside an ambient named n ; dually, $\text{out } n$ allows to exit an ambient n ; $\text{open } n$ dissolves an ambient n .

A process is the void process 0 , a process $n[P]$ obtained by wrapping P in an ambient n , a *sequential* process $M.P$, the parallel composition of two processes $P|Q$, the recursive process $\text{rec } X.P$ or a process variable X .

Definition 8 (Structural equivalence). *The semantics of the Ambient calculus relies on the structural equivalence defined by the following rules:*

1. **The parallel operator** $|-$ is associative, commutative and 0 is its identity;

$$\frac{P \equiv Q}{P \equiv Q} \quad \frac{P \equiv Q}{P \equiv Q}$$
2.
$$\frac{M.P \equiv M.Q}{M.P \equiv M.Q} \quad \frac{n[P] \equiv n[Q]}{n[P] \equiv n[Q]}$$
3. $\text{rec } X.P \equiv \text{rec } Y.P\{Y/X\}$, if $Y \notin \text{fv}(P)$;
4. $\text{rec } X.P \equiv P\{\text{rec } X.P/X\}$.

The usual algebraic properties of the parallel composition and the 0 process are assumed (rule 1); rule 2 guarantees that structural equivalence is preserved by capabilities and ambient processes; the process variable X is bound in $\text{rec } X.P$ and may be renamed (rule 3); finally, rule 4 is the analogous of the usual structural rule for replication (namely, $!P \equiv P \mid !P$).

Definition 9 (Reduction Semantics). *The reduction relation $\rightarrow \subseteq \text{Proc} \times \text{Proc}$ is the relation inductively generated by the axioms and rules in table 3 and closed under the structural equivalence given in Definition 8:*

Table 3. Ambient calculus reduction relation

$$\begin{array}{c}
m[n[\text{out } m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R] \\
n[\text{in } m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R] \\
\text{open } n.P \mid n[Q] \rightarrow P \mid Q \\
\frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R} \quad \frac{P \rightarrow Q}{n[P] \rightarrow n[Q]}
\end{array}$$

The first two axioms in Table 3 state that an ambient n can be driven by a sequential process inside it to exit the wrapping ambient ($\text{out } m.P$) or to enter a parallel ambient m ($\text{in } m.P$). The third axiom is relative to the $\text{open } n$ capability: an ambient may be dissolved by an external process. Note that all the capabilities are “asynchronous”, in the sense that the only condition under which they can be fired is the presence of a particular ambient.

3.2 Graph Representation of Ambient Calculus

We now show how it is possible to translate the Ambient calculus in our graph synchronization framework maintaining the semantics of processes.

Definition 10 (Translation).

$$\begin{aligned}
\llbracket 0 \rrbracket_x &= x \vdash nil \\
\llbracket n[P] \rrbracket_x &= x \vdash \nu y.(G \mid n(y, x)), & \text{if } y \neq x \wedge \llbracket P \rrbracket_y = y \vdash G \\
\llbracket M.P \rrbracket_x &= x \vdash L_{M.P}(x) \\
\llbracket P_1 \mid P_2 \rrbracket_x &= x \vdash G_1 \mid G_2, & \text{if } \llbracket P_i \rrbracket_x = x \vdash G_i, \text{ where } i = 1, 2 \\
\llbracket rec X.P \rrbracket_x &= \llbracket P[rec X.P/X] \rrbracket_x
\end{aligned}$$

Definition 10 introduces the mapping function $\llbracket P \rrbracket_x$ that returns a graph whose only free node x corresponds to the root of the ambient process P .

In the above translation, sequential processes $M.P$ are directly represented by edges labelled by $M.P$. While this introduces an infinite number of labels, it is easy to see that only a finite number of them (and of the corresponding activity rules defined below) is needed to derive all computations of any particular ambient.

The graph associated to the 0 process is an isolated node. The graph of $n[P]$ with free node x is obtained by constructing the graph of P on node y , attaching it to the graph $n(y, x)$ and restricting y ; note that the ambient name n is interpreted as an edge from y to x labelled n . Ambient names N and sequential processes are the only edge labels.

The parallel composition $P_1 \mid P_2$ is obtained by making the graph of P_1 and P_2 to share their root node x ; finally, recursive processes are unfolded first¹.

The given translation is injective but not surjective. However, the graphs $\llbracket P \rrbracket_x$ in the image of the translation function can be characterized as follows.

Definition 11 (Ambient graphs). *An ambient graph is a graph labeled on $LE = \{L_{M.P} \mid M.P \in Proc \text{ is sequential}\} \cup N$ which*

1. *is acyclic;*
2. *every node has at most one outgoing edge labelled in N ;*
3. *there is one root node with no outgoing edges.*

Theorem 1. $\llbracket - \rrbracket_x$ *is a bijection on ambient graphs.*

We now define the productions of our version of the Ambient calculus. There are two kinds of productions: *activity productions*, relative to sequential processes, and *coordination productions* that corresponds to ambients.

Definition 12 (Activity productions). *The activity productions have the following form.*

$$\frac{\llbracket L_{M.P} \rrbracket \longrightarrow \bullet_x^{\overline{M}}}{\Longrightarrow G \quad x \vdash L_{M.P}(x) \xrightarrow{\{(x, \overline{M}, \langle \rangle)\}} \llbracket P \rrbracket_x}$$

¹ Note that the $\llbracket - \rrbracket_x$ is well defined because recursion variables are guarded by capabilities.

Activity productions determine the actions that sequential processes are able to perform. In our approach, sequential processes become edge labels: when an action is performed, an edge labelled by $M.P$ is rewritten as the graph corresponding to P .

The complementary actions to synchronize the activity productions must be offered by ambients; more precisely, ambients must signal their existence emitting the complementary actions on their attaching nodes and, in this manner, performing the correct synchronized steps.

Definition 13 (Coordination productions). *Coordination productions are as follows.*

$$\begin{array}{c}
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{a}} & y \\
 \bullet & & \bullet \\
 \text{(open)} & & \text{open } a
 \end{array}
 \quad \xRightarrow{x \mapsto y} \quad
 \begin{array}{c}
 y \\
 \bullet
 \end{array}
 \\
 x, y \vdash a(x, y) \xrightarrow{\{(y, \text{open } a, \langle \rangle)\} [x \mapsto y]} y \vdash \text{nil}
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{b}} & y \\
 \bullet & & \bullet \\
 \text{(input1)} & & \text{in } a \quad \text{input } a, z
 \end{array}
 & \xRightarrow{\quad} &
 \begin{array}{ccc}
 & & y \\
 & & \bullet \\
 x & \xrightarrow{\boxed{b}} & \bullet \\
 \bullet & & \bullet \\
 & & z
 \end{array}
 \\
 x, y \vdash b(x, y) \xrightarrow{\{(x, \text{in } a, \langle \rangle), (y, \overline{\text{input } a}, \langle z \rangle)\}} x, y, z \vdash b(x, z)
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{a}} & y \\
 \bullet & & \bullet \\
 \text{(input2)} & & \text{input } a, x
 \end{array}
 & \xRightarrow{\quad} &
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{a}} & y \\
 \bullet & & \bullet
 \end{array}
 \\
 x, y \vdash a(x, y) \xrightarrow{\{(y, \text{input } a, \langle x \rangle)\}} x, y \vdash a(x, y)
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{b}} & y \\
 \bullet & & \bullet \\
 \text{(output1)} & & \text{out } a \quad \text{output } a, z
 \end{array}
 & \xRightarrow{\quad} &
 \begin{array}{ccc}
 & & y \\
 & & \bullet \\
 x & \xrightarrow{\boxed{b}} & \bullet \\
 \bullet & & \bullet \\
 & & z
 \end{array}
 \\
 x, y \vdash b(x, y) \xrightarrow{\{(x, \text{out } a, \langle \rangle), (y, \overline{\text{output } a}, \langle z \rangle)\}} x, y, z \vdash b(x, z)
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{a}} & y \\
 \bullet & & \bullet \\
 \text{(output2)} & & \text{output } a, y
 \end{array}
 & \xRightarrow{\quad} &
 \begin{array}{ccc}
 x & \xrightarrow{\boxed{a}} & y \\
 \bullet & & \bullet
 \end{array}
 \\
 x, y \vdash a(x, y) \xrightarrow{\{(x, \text{output } a, \langle y \rangle)\}} x, y \vdash a(x, y)
 \end{array}$$

For every production, we give both the sequent and its graphical representation. In the latter, left and right members of a production are drawn in the style of Definition 10, but without restricted nodes. When $(x, \mu, \langle y \rangle) \in \Gamma$, node x in the right member is labeled by x, μ . Coordination productions define the complementary actions that ambients must perform in order to synchronize themselves with sequential processes.

The *(open)* production states that if the ambient a has a parallel process that wants to open it, then the edge corresponding to a disappears and x is fused with y .

Production *(input1)* asserts that is a process inside b wants to drive b in an ambient a , then the destination of b will become the new node z . On the other hand, production *(input2)* controls the entrance of an external process in a : this production simply passes the source x of a to the entering process.

Analogously to the input productions, *(output1)* and *(output2)* take care of the output action. We remark that *(output1)* acts quite similarly to *(input1)*.

Definition 14 (Basic transition). A transition $\Gamma \vdash G \xrightarrow{\Lambda, \pi} \phi \vdash G'$ is basic if:

- π is the identity function on Γ ;
- its proofs uses exactly one instance of either *(open)* or *(input1)* or *(output1)*;
- Λ is either a singleton $\{(x, \tau, \langle \rangle)\}$ or it is empty.

Theorem 2. For all ambient processes $P, Q \in Proc$:

- if $P \rightarrow Q$ then $\llbracket P \rrbracket_x \xrightarrow{\Lambda, id} \llbracket Q \rrbracket_x$ and either $\Lambda = \emptyset$ or $\Lambda = \{(x, \tau, \langle \rangle)\}$;
- if $\llbracket P \rrbracket_x \xrightarrow{\Lambda, \pi} \phi \vdash G$ is a basic transition, then $\phi \vdash G = \llbracket Q \rrbracket_x$ and $P \rightarrow Q$.

Proof (sketch): The proof of the theorem is based on the fact that if a basic derivation $\llbracket P \rrbracket_x \xrightarrow{\Lambda, id}$ exists, then it is possible to derive the same transition by

1. applying instances of the *(par)* rule to a suitable set of productions;
2. applying the *(merge)* rule in such a way that the graph relative to P without restrictions is obtained;
3. restricting by means of rule *(res)* all the nodes that are not the root node of P .

It is easy to note that all the reductions that do not take place at the top level of P correspond to basic transitions of the graph whose Λ is empty, while the transitions that involve subprocesses of P at the top level have $\Lambda = \{(x, \tau, \langle \rangle)\}$.

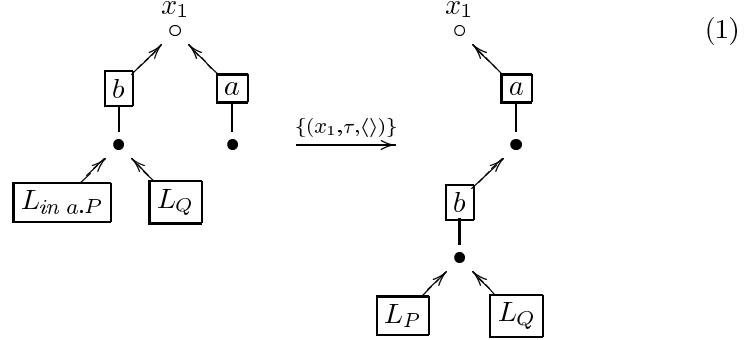
3.3 Example

As an example we show the correspondence between an Ambient calculus reduction and the corresponding graph transition. Let us consider the ambient reduction

$$b[in\ a.P \mid Q] \mid a[0] \rightarrow a[b[P \mid Q]]$$

where P and Q are sequential processes.

Following Definition 10 and Theorem 2, we should obtain



The picture on the left is the graphical representation of $[[b[in a.P \mid Q] \mid a[0]]]_{x_1}$, while the rightmost picture is $[[a[b[P \mid Q]]]]_{x_1}$ (we represent the restricted nodes with \bullet and the free nodes with \circ).

The steps described in the proof sketch of Theorem 2 guide us in applying the productions (activity and coordination) and the inference rules of Table 2 in order to construct a proof for transition (1).

First (step 1) we decompose the graph in its elementary edges and determine the productions that correspond to the elementary components of the transition.

$$x_1, y_1 \vdash b(y_1, x_1) \xrightarrow{\left\{ \begin{array}{l} (x_1, \overline{input\ a}, \langle z_1 \rangle), \\ (y_1, in\ a, \langle \rangle) \end{array} \right\}, id} x_1, y_1, z_1 \vdash b(y_1, z_1) \quad (2)$$

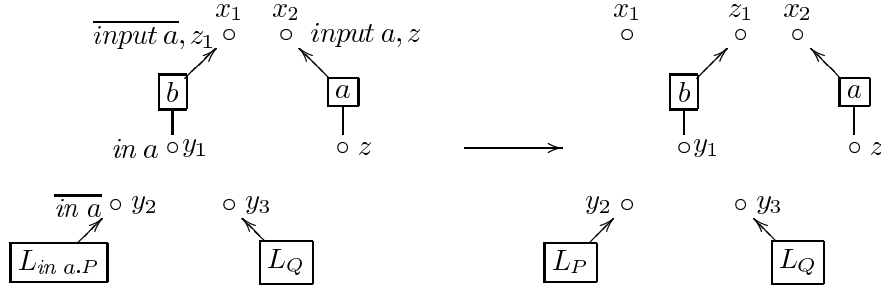
$$y_2 \vdash L_{in\ a.P}(y_2) \xrightarrow{\{(y_2, \overline{in\ a}, \langle \rangle)\}, id} y_2 \vdash L_P(y_2) \quad (3)$$

$$x_2, z \vdash a(z, x_2) \xrightarrow{\{(x_2, input\ a, \langle z \rangle)\}, id} x_2, z \vdash a(z, x_2) \quad (4)$$

$$y_3 \vdash L_Q(y_3) \xrightarrow{\emptyset, id} y_3 \vdash L_Q(y_3) \quad (5)$$

Transitions (2) and (4) are instances of the coordination productions (*input1*) and (*input2*), respectively; transition (3) is the activity production relative to *in a.P* and transition (5) is the identity transition that leaves *L_Q* idle.

The graphical representation is:



The previous graph represents the transition obtained by applying the (*par*) rule to the productions (2), (3), (4) and (5). Let

$$\begin{aligned} G_1 &= b(y_1, x_1) \mid a(z, x_2) \mid L_{in\ a.P}(y_2) \mid L_Q(y_3) \\ G_2 &= b(y_1, z_1) \mid a(z, x_2) \mid L_P(y_2) \mid L_Q(y_3) \\ \Gamma &= \{x_1, x_2, y_1, y_2, y_3, z\} \end{aligned}$$

then, in terms of sequents we have:

$$\Gamma \vdash G_1 \xrightarrow{\left\{ \begin{array}{l} (x_1, \overline{input\ a}, \langle z_1 \rangle), \\ (x_2, \overline{input\ a}, \langle z \rangle) \\ (y_1, \overline{in\ a}, \langle \rangle) \\ (y_2, \overline{in\ a}, \langle \rangle) \end{array} \right\}, id} \Gamma, z_1 \vdash G_2 \quad (6)$$

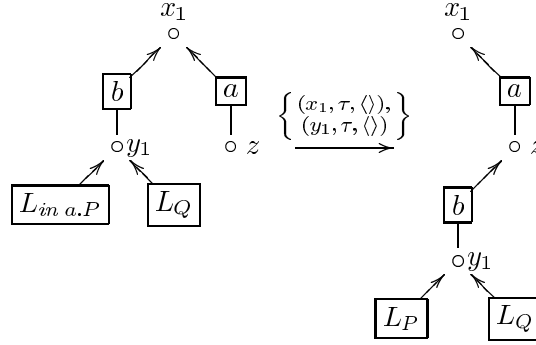
The application of the merge rule (step 2) provides the fusion of the nodes in order to obtain a graph of the same shape of the ambient process but without restricted nodes. Referring to the rule (*merge*), let σ the function that behaves as the identity on all nodes different from x_2 , y_2 and y_3 and

$$\sigma : \begin{cases} x_2 \mapsto x_1 \\ y_2 \mapsto y_1 \\ y_3 \mapsto y_1 \end{cases}$$

that determines $\Lambda' = \{(x_1, \tau, \langle \rangle), (y_1, \tau, \langle \rangle)\}$ and $\rho : z_1 \mapsto z$. The rule (*merge*) may be applied to transition (6) obtaining the transition

$$x_1, y_1, z \vdash \sigma(G_1) \xrightarrow{\left\{ \begin{array}{l} (x_1, \tau, \langle \rangle), \\ (y_1, \tau, \langle \rangle) \end{array} \right\}, id} x_1, y_1, z \vdash \rho(\sigma(G_2))$$

that is graphically represented as



We remark that the above transition requires a synchronization involving three edges and two nodes: the edges relative to *in a.P* and *b* that synchronize on node y_1 , and the edges relative to ambients *b* and *a* that synchronize on node x_1 . This makes clear that the *in* capability of ambients requires to synchronize three components (the *out* capability is analogous).

Finally, two applications of the (*res*) rule (step 3) are needed in order to restrict nodes z and y_1 . This concludes the proof of the transition.

4 Conclusion

In the paper we presented a simple LTS semantics of Cardelli and Gordon's Ambient calculus exploiting a graphical model based on edge replacement and local synchronization with mobility. While the correspondence with the original operational semantics of ambients is shown for a restricted class of graphs (the ambient graphs), it is also conceivable to lift this limitation and to allow all graphs on the same edge labels. Coordination productions should be exactly the same, while activity productions should be allowed to rewrite an edge into any such graph. The resulting calculus should allow programmable ambient mobility on any graphical model of internetworks, providing a more realistic description of real systems.

The work presented here is still at an initial stage. The labeled transition system defined by the logical sequents in the paper automatically provides an abstract semantics of ambients under the usual definition of bisimilarity. Since all nodes in ambient graphs are restricted except for the root, interactions can only be observed there. This should respect the intuition of abstract semantics of ordinary ambients, where barb observation and ambient composition are via the root. However we did not yet study the relation of our abstract semantics with Cardelli and Gordon's, and we do not know if it is a congruence, i.e. if it is respected by our operations of composition and restriction of graphs. Finally, we would like to experiment with network reconfiguration techniques more general than ambients, but still realistic for actual internetworks, taking advantage of our general approach.

References

1. Luca Cardelli and Andrew D. Gordon. A commitment relation for the ambient calculus. Manuscript. [2](#)
2. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *TCS: Theoretical Computer Science*, 240, 2000. [1](#), [8](#)
3. Ilaria Castellani and Ugo Montanari. Graph Grammars for Distributed Systems. In Hartmut Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors, *Proc. 2nd Int. Workshop on Graph-Grammars and Their Application to Computer Science*, volume 153 of *Lecture Notes in Computer Science*, pages 20–38. Springer-Verlag, 1983. [3](#)
4. Rocco De Nicola, Gianluigi Ferrari, and Rosario Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998. Special Issue: Mobility and Network Aware Computing. [1](#)
5. P. Degano and Ugo Montanari. A model of distributed systems based of graph rewriting. *Journal of the ACM*, 34:411–449, 1987. [3](#)
6. Gianluigi Ferrari, Carlo Montangero, Laura Semini, and Simone Semprini. Mobile agents coordination in *Mob_{adt}*. In Antonio Porto and Gruiia-Catalin Roman, editors, *Coordination Languages and Models*, volume 1906 of *LNCS*. Springer Verlag, 2000. [1](#)

7. Cedric Fournet and George Gonthier. The reflexive CHAM and the join-calculus. In *Conference Record of POPL '96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, 21–24 January 1996. [1](#)
8. Dan Hirsch, Paola Inverardi, and Ugo Montanari. Reconfiguration of software architecture styles with name mobility. In Antonio Porto and Gruiia-Catalin Roman, editors, *Coordination 2000*, volume 1906 of *LNCS*, pages 148–163. Springer Verlag, 2000. [3](#), [6](#)
9. Dan Hirsh and Ugo Montanari. Synchronized hyperedge replacement with name mobility. In *To appear in CONCUR01*, 2001. [3](#), [5](#), [6](#)
10. Barbara Koenig and Ugo Montanari. Observational equivalence for synchronized graph rewriting. In *Proc. TACS'01*, LNCS. Springer Verlag, 2001. To appear. [3](#), [6](#)
11. Ugo Montanari and Francesca Rossi. Graph rewriting and constraint solving for modelling distributed systems with synchronization. In P. Ciancarini and C. Hankin, editors, *Proceedings of the First International Conference COORDINATION '96, Cesena, Italy*, volume 1061 of *LNCS*. Springer Verlag, April 1996. [3](#), [5](#)
12. Gian Pietro Picco, Amy L. Murphy, and Gruiia-Catalin Roman. LIME: Linda Meets Mobility. In D. Garlan, editor, *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, pages 368–377, Los Angeles, CA, USA, May 1999. ACM Press. Also available as Technical Report WUCS-98-21, July 1998, Washington University in St. Louis, MO, USA. [1](#)
13. Gruiia-Catalin Roman, Peter J. McCann, and J. Y. Plunn. Mobile UNITY: Reasoning and specification in mobile computing. *ACM Transactions on Software Engineering and Methodology*, 6(3):250–282, July 1997. [1](#)
14. Peter Sewell. From rewrite rules to bisimulation congruences. *Lecture Notes in Computer Science*, 1466, 1998. [2](#)
15. Bjorn Victor and Joachim Parrow. Concurrent constraints in the fusion calculus. *Lecture Notes in Computer Science*, 1443, 1998. [6](#)