Electronic Notes in Theoretical Computer Science 45 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume45.html 18 pages

A Concurrent Graph Semantics for Mobile Ambients [1]

Fabio Gadducci and Ugo Montanari [2]

Dipartimento di Informatica
Università di Pisa
Pisa, Italy

Abstract

We present an encoding for finite processes of the mobile ambients calculus into term graphs, proving its soundness and completeness with respect to the original, interleaving operational semantics. With respect to most of the other approaches for the graphical implementation of calculi with name mobility, our term graphs are unstructured (that is, non hierarchical), thus avoiding any "encapsulation" of processes. The implication is twofold. First of all, it allows for the reuse of standard graph rewriting theory and tools for simulating the reduction semantics. More importantly, it allows for the simultaneous execution of independent reductions, which are nested inside ambients, thus offering a concurrent semantics for the calculus.

Key words: concurrent graph rewriting, graphical encoding of process calculi, mobile ambients, reduction semantics.

1 Introduction

After the development of so-called optimal implementation of λ-calculus, many authors proposed graphical presentations for calculi with name mobility, in particular for the π-calculus [24]. These proposals usually introduce a syntactical notation for graphs, then they map processes into graphs *via* that notation. With a few exceptions [13,27], the resulting graphical structures are eminently hierarchical (that is, roughly, each node/edge/label is itself a structured entity, and possibly a graph), thus forcing the development of ad-hoc mechanisms for graph rewriting, in order to simulate process reduction.

[1] Research partly supported by the EC TMR Network *General Theory of Graph Transformation Systems* (GETGRATS); by the EC Esprit WG *Applications of Graph Transformations* (APPLIGRAPH); and by the Italian MURST Project *Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi* (TOSCA).

[2] Email: gadducci@di.unipi.it, ugo@di.unipi.it

In this paper we present instead a general proposal for mapping processes of calculi with name mobility into unstructured, non-hierarchical graphs. As the main example we chose mobile ambients [6], partly for its rising popularity in the community, while still lacking an analysis of its concurrency features; and partly because the complex name handling presented by its reduction rules highlights the power of our framework.

In fact, we believe that the intuitive appeal of non-hierarchical graphs, and the local nature of the associated rewriting mechanism, may help cast some light on the distributed features of the calculus. To this end, our first step is to prove the soundness and correctness of our encoding of processes into graphs, in the sense that two processes are structurally equivalent if and only if the corresponding graphs are isomorphic. Our second step is to prove that the encoding is faithful with respect to the reduction semantics, in the sense that standard graph rewriting techniques may now be used to simulate reduction steps on processes by sequences of rewrites on their encodings.

One of the additional advantages of formulating the reduction semantics of mobile ambients in terms of graph rewriting is the existence of a well-developed concurrent semantics [1], which extends the concurrent semantics of Petri nets and which allows to derive graph processes, event structures and prime algebraic domains from graph transformation systems. A concurrent semantics puts an upper limit to the amount of parallelism that is intrinsic in the reductions, and moreover it allows to derive causality links between reduction steps, which can be useful in better understanding the behaviour of a process, e.g. with respect to security and non-interference.

The paper has the following structure: In Section 2 we recall the mobile ambients calculus, and we discuss two alternative reduction semantics. In Section 3 we introduce a set-theoretical presentation for (ranked term) graphs, and we define two operations on them, namely *sequential* and *parallel composition* [7,8]. These operations are used in Section 4 to formulate our encoding for processes of the mobile ambient calculus, which is then proved to be sound and complete with respect to structural congruence. Finally, in Section 5 we recall the basic tools of graph rewriting, according to the DPO approach, and we show how four simple graph rewriting rules allow for simulating the reduction semantics of the mobile ambients calculus. We then argue how the information on causal dependencies between rewriting steps offered by the concurrent semantics of graph rewriting may be used for detecting *interferences* among process reductions, according to the taxonomy proposed in [22]. We close the paper with a few remarks, concerning the relevance of mapping processes into unstructured graphs from the point of view of parallelism; the generality of the approach, and its relationship with ongoing work on the graphical presentation of algebraic formalisms; and finally, the way to extend our results, in order to handle recursive processes.

$$P = Q \quad \text{for } P, Q \ \alpha\text{-convertible;}$$
$$P \mid Q = Q \mid P, \qquad P \mid (Q \mid R) = (P \mid Q) \mid R, \qquad P \mid \mathtt{0} = P;$$
$$(\nu n)(\nu m)P = (\nu m)(\nu n)P \qquad (\nu n)(P \mid Q) = P \mid (\nu n)Q \quad \text{for } n \notin \mathit{fn}(P).$$
$$(\nu n)m[P] = m[(\nu n)P] \quad \text{for } n \neq m$$

Fig. 1. The set of axioms without deadlock detection

$$(\nu n)\mathtt{0} = \mathtt{0}$$

Fig. 2. The additional axiom for deadlock detection

2 Structural congruences for mobile ambients

This section shortly introduces the finite, communication-free fragment of the mobile ambients calculus, its structural equivalence and the associated reduction semantics. In addition, we describe two alternative structural equivalences for the calculus, proving that the associated reduction semantics are in fact "coincident", in a way to be made precise later on, to the original semantics.

2.1 The original calculus

Definition 2.1 (processes) *Let $\mathcal{N}$ be a set of atomic* names*, ranged over by $m, n, o, \ldots$. A* process *is a term generated by the following syntax*

$$P ::= \mathtt{0},\ n[P],\ M.P,\ (\nu n)P,\ P_1 \mid P_2$$

for the set of capabilities

$$M ::= \mathit{in}\, n,\ \mathit{out}\, n,\ \mathit{open}\, n.$$

We let $P, Q, R, \ldots$ range over the set Proc of processes.

We assume the standard definitions for the set of free names of a process P, denoted by $\mathit{fn}(P)$. Similarly for α-convertibility, with respect to the *restriction* operators (νn). Using these definitions, the dynamic behaviour of a process P is described as a relation over *abstract processes*, i.e., a relation obtained by closing a set of basic rules under structural congruence.

Definition 2.2 (reduction semantics) *The* reduction relation *for processes is the relation $R_m \subseteq \mathit{Proc} \times \mathit{Proc}$, closed under the structural congruence $\cong$ induced by the set of axioms in Figure 1 and Figure 2, inductively generated by the following set of axioms and inference rules*

$$\frac{}{m[n[\mathit{out}\, m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]}$$

$$\frac{}{n[\mathit{in}\, m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]} \qquad \frac{}{\mathit{open}\, n.P \mid n[Q] \to P \mid Q}$$

$$\frac{P \to Q}{(\nu n)P \to (\nu n)Q} \qquad \frac{P \to Q}{P \mid R \to Q \mid R} \qquad \frac{P \to Q}{n[P] \to n[Q]}$$

where $P \to Q$ means that $\langle P, Q \rangle \in R_m$.

$$(\nu n)M.P = M.(\nu n)P \quad \text{for } n \notin fn(M)$$

Fig. 3. The additional axiom for capability floating

2.2 Two alternative structural congruences

An important novelty in calculi with name mobility is the use of structural congruence for presenting the reduction semantics. This is intuitively appealing, since *abstract* processes allows for a simple representation (that is, modulo a suitable equivalence) of the spatial distribution of a system. Many equivalences, though, may be taken into account. Let us denote respectively as $P \rightarrow_d Q$ the reduction relation obtained by closing the inference rules presented in Definition 2.2 with respect to the structural congruence, denoted by $\cong_d$, induced by the set of axioms in Figure 1; and by $P \rightarrow_f Q$ the reduction relation obtained by closing the inference rules presented in Definition 2.2 with respect to the structural congruence, denoted by $\cong_f$, induced by the set of axioms in Figure 1 and Figure 3.

The first equivalence $\cong_d$ is finer than $\cong$, since it just forbids the identification of the deadlocked processes $\mathtt{0}$ and $(\nu n)\mathtt{0}$. Nevertheless, the mapping from abstract processes according to $\cong_d$, into abstract processes according to $\cong$, faithfully preserves the reduction semantics, as stated by next theorem.

Proposition 2.3 (deadlock and reductions) *Let P, Q be processes.* (1) *If $P \rightarrow_d Q$, then $P \rightarrow Q$. Vice versa,* (2) *if $P \rightarrow Q$, then there exists a process R such that $P \rightarrow_d R$ and $Q \cong R$.*

In other terms, the mapping does not add reductions. Sometimes, these kinds of mapping are also called *transition preserving morphisms* [11], a special form of the general notion of *open map* [18]. A similar property is satisfied by the mapping from abstract processes according to $\cong_d$, into abstract processes according to $\cong_f$, adding the distributivity of restriction with respect to capability (that is, letting the restrictions float to the top of a term).

Proposition 2.4 (distributivity and reductions) *Let P, Q be processes.* (1) *If $P \rightarrow_d Q$, then $P \rightarrow_f Q$. Vice versa,* (2) *if $P \rightarrow_f Q$, then there exists a process R such that $P \rightarrow_d R$ and $Q \cong_f R$.*

Our main theorem will present an alternative characterisation of the relation $\rightarrow_f$ by means of graph rewriting techniques.

3 Graphs and term graphs

We open the section recalling the definition of (ranked) term graphs: We refer to [5,7] for a detailed introduction, as well as for a comparison with standard definitions such as [3]. In particular, we assume in the following a chosen signature (Σ, S), for Σ a set of operators, and S a set of sorts, such that the *arity* of an operator in Σ is a pair (ω_s, ω_t), for ω_s, ω_t strings in S^*.

Definition 3.1 (graphs) *A* labelled graph *d (over (Σ, S)) is a five tuple $d = \langle N, E, l, s, t\rangle$, where N, E are the sets of* nodes *and* edges*; l is the pair of* labeling *functions $l_e : E \to \Sigma$, $l_n : N \to S$; $s, t : E \to N^*$ are the* source *and* target *functions; and such that for each edge $e \in dom(l)$, the arity of $l_e(e)$ is $(l_n^*(s(e)), l_n^*(t(e)))$, i.e., each edge preserves the arity of its label.*

With an abuse of notation, in the definition above we let l_n^* denote the extension of the function l_n from nodes to strings of nodes. In the following, we denote the components of a graph d by N_d, E_d, l_d, s_d and t_d.

Definition 3.2 (graph morphisms) *Let d, d' be graphs. A* (graph) morphism *$f : d \to d'$ is a pair of functions $f_n : N_d \to N_{d'}$, $f_e : E_d \to E_{d'}$ that preserves the labeling, source and target functions.*

In order to inductively define an encoding for processes, we need to define some operations over graphs. The first step is to equip them with suitable "handles" for interacting with an environment, built out of other graphs.

Definition 3.3 ((ranked) term graphs) *Let d_r, d_v be graphs with no edges. A (d_r, d_v)-*ranked graph *(a* graph of rank (d_r, d_v)*) is a triple $g = \langle r, d, v\rangle$, for d a graph and $r : d_r \to d$, $v : d_v \to d$ the injective* root *and* variable *morphisms.*

Let g, g' be ranked graphs of the same rank. A ranked graph morphism *$f : g \to g'$ is a graph morphism $f_d : d \to d'$ between the underlying graphs that preserves the root and variable morphisms.*

Two graphs $g = \langle r, d, v\rangle$ and $g' = \langle r', d', v'\rangle$ of the same rank are isomorphic *if there exists a* ranked graph isomorphism *$\phi : g \to g'$. A (d_r, d_v)-ranked term graph G is an isomorphism class of (d_r, d_v)-ranked graphs.*

With an abuse of notation, we sometimes refer to the nodes in the image of the variable (root) morphism as variables (roots, respectively). Moreover, we often use the same symbols of ranked graphs to denote term graphs, so that e.g. $G_{d_v}^{d_r}$ denotes a term graph of rank (d_r, d_v).

Definition 3.4 (sequential and parallel composition) *Let $G_{d_v}^{d_i}$, $H_{d_i}^{d_r}$ be term graphs. Their* sequential composition *is the term graph $G_{d_v}^{d_i}; H_{d_i}^{d_r}$ of rank (d_r, d_v) obtained by first the disjoint union of the graphs underlying G and H, and second the gluing of the roots of G with the corresponding variables of H.*

Let $G_{d_v}^{d_r}$, $H_{d'_v}^{d'_r}$ be term graphs, such that $d_v \cap d'_v = \emptyset$. Their parallel composition *is the term graph $G_{d_v}^{d_r} \otimes H_{d'_v}^{d'_r}$ of rank $(d_r \cup d'_r, d_v \cup d'_v)$ obtained by first the disjoint union of the graphs underlying G and H, and second the gluing of the roots of G with the corresponding roots of H.* [3]

[3] Let $G_{d_v}^{d_i} = \langle r, d, v\rangle$ and $H_{d_i}^{d_r} = \langle r', d', v'\rangle$ be term graphs. Then, $G; H = \langle r'', d'', v''\rangle$, for d'' the disjoint union of d and d', modulo the equivalence on nodes induced by $r(x) = v'(x)$ for all $x \in N_{d_i}$, and $r'' : d_r \to d''$, $v'' : d_v \to d''$ the uniquely induced arrows. Let now $G_{d_v}^{d_r} = \langle r, d, v\rangle$ and $H_{d'_v}^{d'_r} = \langle r', d', v'\rangle$ be term graphs. Then, $G \otimes H = \langle r'', d'', v''\rangle$, for d'' the disjoint union of d and d', modulo the equivalence on nodes induced by $r(x) = r'(x)$ for all $x \in N_{d_r} \cap N_{d'_r}$, and $r'' : d_r \cup d'_r \to d''$, $v'' : d_v \cup d'_v \to d''$ the uniquely induced arrows.

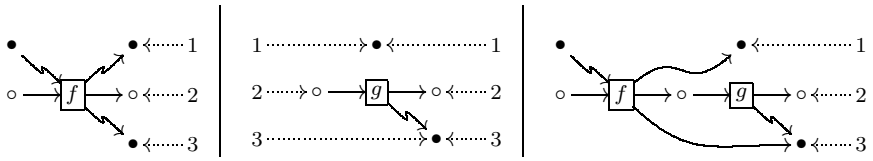

Fig. 4. Two term graphs, and their sequential composition

Note that the two operations are defined on "concrete" graphs. Nevertheless, the result is clearly independent of the choice of the representative, and it implies that both parallel and sequential composition are associative.

Example 3.5 (sequential composition) *Let us consider the signature* (Σ_e, S_e)*, for* $S_e = \{s_1, s_2\}$ *and* $\Sigma_e = \{f : s_1 s_2 \to s_1 s_2 s_1, g : s_2 \to s_2 s_1\}$*. Two term graphs, built out of the signature* (Σ_e, S_e)*, are shown in Figure 4. The nodes in the domain of the root (variable) morphism are depicted as a vertical sequence on the right (left, respectively); edges are represented by their label, from where arrows pointing to the target nodes leave, and to where the arrows from the source node arrive. The root and variable morphisms are represented by dotted arrows, directed from right-to-left and left-to-right, respectively.*

The term graph on the left has rank $(\{1, 2, 3\}, \emptyset)$*, five nodes and one edge (labelled by* f*); the term graph on the middle has rank* $(\{1, 2, 3\}, \{1, 2, 3\})$*, four nodes and one edge (labelled by* g*). For graphical convenience, in the underlying graph the nodes of sort* s_1 *are denoted by* $\bullet$*, those of sort* s_2 *by* $\circ$*.*

Sequential composition of term graphs is performed by matching the roots of the first graph with the variables of the second one, as shown by the term graph on the right: It has rank $(\{1, 2, 3)\}, \emptyset)$*, six nodes and two edges, and it is obtained by sequentially composing the other two.*

A *(term graph) expression* is a term over the signature containing all ranked term graphs as constants, and parallel and sequential composition as binary operators. An expression is *well-formed* if all occurrences of both parallel and sequential composition are defined for the rank of the argument subexpressions, according to Definition 3.4; the *rank* of an expression is then computed inductively from the rank of the term graphs appearing in it, and its *value* is the term graph obtained by evaluating all operators in it.

4 Channels as wires: from processes to term graphs

The first step in our implementation is to encode processes into term graphs, built out of a suitable signature (Σ_m, S_m), and proving that the encoding preserves structural convertibility. Then, standard graph rewriting techniques are used for simulating the reduction mechanism.

The set of sorts S_m contains the elements s_p and s_a. The first symbol is reminiscent of the word *process*, since the elements of sort s_p can be considered as processes reached by a transition. The second sort, s_a, is reminiscent of *ambient*, and the elements of this sort correspond to names of the calculus.

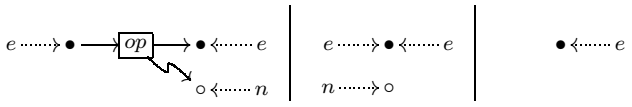

Fig. 5. Term graphs op_n (for $op \in \{amb, in, open, out\}$), ν_n and $\mathtt{0}$.

Fig. 6. Term graphs op (for $op \in \{go, idle\}$), new_n e id_n.

The operators are $\{in : s_p \to s_p s_a, out : s_p \to s_p s_a, open : s_p \to s_p s_a\} \cup \{amb : s_p \to s_p s_a\} \cup \{go : \lambda \to s_p, idle : \lambda \to s_p\}$. The elements of the first set simulate the capabilities of the calculus; the *amb* operator simulates ambients. Note that there is no operator for simulating name restriction; instead, the operators *go* and *idle* are syntactical devices for detecting the status of those nodes in the source of an edge labeled *amb*, thus avoiding to perform any reduction below the outermost capability operator, as shown in Section 5.

The second step is the characterisation of 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. Thus, let us consider a name $e \notin \mathcal{N}$: Our choice is depicted in Figure 5 and Figure 6.

Definition 4.1 (encoding for processes) *Let P be a process, and let Γ be a set of names, such that $fn(P) \subseteq \Gamma$. The* encoding $[\![P]\!]^{go}_{\Gamma}$ *maps a process P into a term graph, as defined below by structural induction,*

$$
\begin{aligned}
[\![P]\!]^{go}_{\Gamma} &= [\![P]\!]_{\Gamma} \otimes go \\
[\![\mathtt{0}]\!]_{\Gamma} &= \mathtt{0} \otimes (\textstyle\bigotimes_{o \in \Gamma} new_o) \\
[\![n[P]]\!]_{\Gamma} &= ([\![P]\!]_{\Gamma} \otimes idle); (amb_n \otimes (\textstyle\bigotimes_{o \in \Gamma} id_o)) \\
[\![M.P]\!]_{\Gamma} &= [\![P]\!]_{\Gamma}; (M_n \otimes (\textstyle\bigotimes_{o \in \Gamma} id_o)) \text{ for } M \text{ capability with } fn(M) = \{n\} \\
[\![(\nu n)P]\!]_{\Gamma} &= [\![P\{^m/_n\}]\!]_{\{m\} \cup \Gamma}; (\nu_m \otimes (\textstyle\bigotimes_{o \in \Gamma} id_o)) \text{ for name } m \notin \Gamma \\
[\![P \mid Q]\!]_{\Gamma} &= [\![P]\!]_{\Gamma} \otimes [\![Q]\!]_{\Gamma}
\end{aligned}
$$

where we assume the standard definition for name substitution.

Thus, the mapping prefixes the term graph $[\![P]\!]_{\Gamma}$ with the occurrence of a "ready" tag, the *go* operator: It will denote an activating point for reduction.

The mapping is well-defined, in the sense that the result is independent of the choice of the name m in the rule for restriction; moreover, given a set of names Γ, the encoding $[\![P]\!]^{go}_{\Gamma}$ of a process P is a term graph of rank $(\{e\} \cup \Gamma, \emptyset)$.

Example 4.2 (a graphical view of firewalls) *We present the implementation of a firewall access, as proposed by Cardelli and Gordon [6]. First, some graphical conventions. The encoding of a process P is a term graph $G = [\![P]\!]_{\{k\}}$ of rank $(\{e, k\}, \emptyset)$: We represent it by circling the expression, from where two dashed arrows leave, directed to the roots of G (hence, to the nodes of G pointed by e and k, respectively). The term graph $[\![k[open\, k.Q]]\!]^{go}_{\{k\}}$ is shown in Figure 7.*

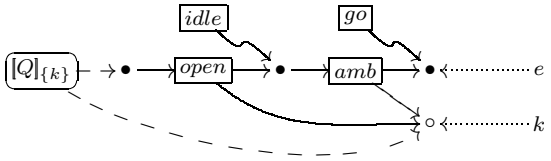

Fig. 7. Term graph for $Agent(Q) = k[open\, k.Q]$.

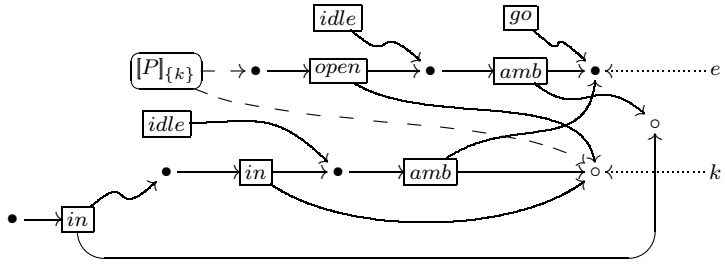

Fig. 8. Term graph for $Firewall(P) = (\nu w)(w[open\, k.P] \mid k[in\, k.in\, w.\mathtt{0}])$.

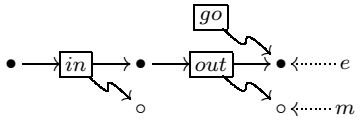

Fig. 9. Term graph encoding for both $(\nu n)out\, m.in\, n.\mathtt{0}$ and $out\, m.(\nu n)in\, n.\mathtt{0}$.

The process $(\nu w)(w[open\, k.P] \mid k[in\, k.in\, w.\mathtt{0}])$*, simulating a firewall, is instead implemented by the ranked term graph in Figure 8.*

The mapping $[\![-]\!]^{go}_{\Gamma}$ is not surjective, because there are term graphs of rank $(\{e\} \cup \Gamma, \emptyset)$ that are not the image of any process; nevertheless, our encoding is sound and complete, as stated by the proposition below.

Proposition 4.3 *Let P, Q be processes, and let Γ be a set of names, such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then, $P \cong_f Q$ if and only if $[\![P]\!]^{go}_{\Gamma} = [\![Q]\!]^{go}_{\Gamma}$.*

Our encoding is thus sound and complete with respect to equivalence $\cong_f$. It is easy to see e.g. that the processes $(\nu n)out\, m.in\, n.\mathtt{0}$ and $out\, m.(\nu n)in\, n.\mathtt{0}$, for $n \neq m$, are mapped to the same term graph, represented in Figure 9.

5 Reductions as graph rewrites

We open the section recalling the basic tools of the double-pushout (DPO) approach to graph rewriting, as presented in [9,10], and introducing a mild generalisation of its well-understood *process semantics* [1]. We then provide a graph rewriting system $\mathcal{R}_m$ for modeling the reduction semantics of mobile ambients. Finally, we discuss the concurrent features of the rewriting system $\mathcal{R}_m$, as captured by the process semantics, arguing that they enhance the analysis of the causal dependencies among the possible reductions performed by a mobile ambient process, with respect to the original interleaving semantics.

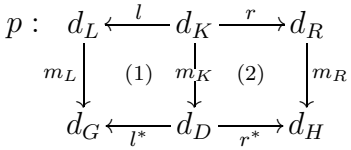

Fig. 10. A DPO direct derivation

5.1 Tools of DPO *graph rewriting*

Definition 5.1 (graph production and derivation) *A* graph production $p : \sigma$ *is composed of a* production name p *and of a span of graph morphisms* $\sigma = (d_L \xleftarrow{l} d_K \xrightarrow{r} d_R)$. *A* graph transformation system *(or* GTS*)* $\mathcal{G}$ *is a set of productions, all with different names. Thus, when appropriate, we denote a production* $p : \sigma$ *using only its name* p.

A graph production $p : (d_L \xleftarrow{l} d_K \xrightarrow{r} d_R)$ *is* injective *if* l *is injective. A graph transformation system* $\mathcal{G}$ *is* injective *if all its productions are so.*

A double-pushout *diagram is like the diagram depicted in Figure 10, where top and bottom are spans and (1) and (2) are pushout squares in the category* $\mathbf{G}_{\Sigma,S}$ *of graphs and graph morphisms (over the signature* (Σ, S)*). Given a production* $p : (d_L \xleftarrow{l} d_K \xrightarrow{r} d_R)$*, a* direct derivation *from* d_G *to* d_H *via production* p *and triple* $m = \langle m_L, m_K, m_R \rangle$ *is denoted by* $d_G \overset{p/m}{\Longrightarrow} d_H$.
A derivation *(of length* n*)* ρ *in a* GTS $\mathcal{G}$ *is a finite sequence of direct derivations* $d_{G_0} \overset{p_1/m_1}{\Longrightarrow} \ldots \overset{p_n/m_n}{\Longrightarrow} d_{G_n}$ *where* $p_1, \ldots, p_n$ *are productions of* $\mathcal{G}$.

Operationally, the application of a production p to a graph d_G consists of three steps. First, the *match* $m_L : d_L \to d_G$ is chosen, providing an occurrence of d_L in d_G. Then, all objects of G matched by $d_L - l(d_K)$ are removed, leading to the *context graph* d_D. Finally, the objects of $d_R - r(d_K)$ are added to d_D, obtaining the derived graph d_H.

The role of the *interface* graph d_K in a rule is to characterise the elements of the graph to be rewritten that are read but not consumed by a direct derivation. Such a distinction is important when considering *concurrent* derivations, possibly defined as an equivalence class of concrete derivations up-to so-called *shift equivalence* [9], identifying (as for the analogous, better-known *permutation equivalence* of λ-calculus) those derivations which differ only for the scheduling of independent steps. Roughly, the equivalence states the interchangeability of two direct derivations $d_1 \Longrightarrow d_2 \Longrightarrow d_3$ if they act either on disjoint parts of d_1, or on parts that are in the image of the interface graphs.

A more concrete, yet equivalent notion of abstract derivation for a GTS is obtained by means of the so-called *process semantics*. As for the similar notion on Petri nets [15], a graph process represents a description for a derivation that abstracts from the ordering of causally unrelated steps (as it is the case for shift equivalence), and that offers at the same time a concrete representative for a class of equivalent derivations. The definition below slightly generalises [1].

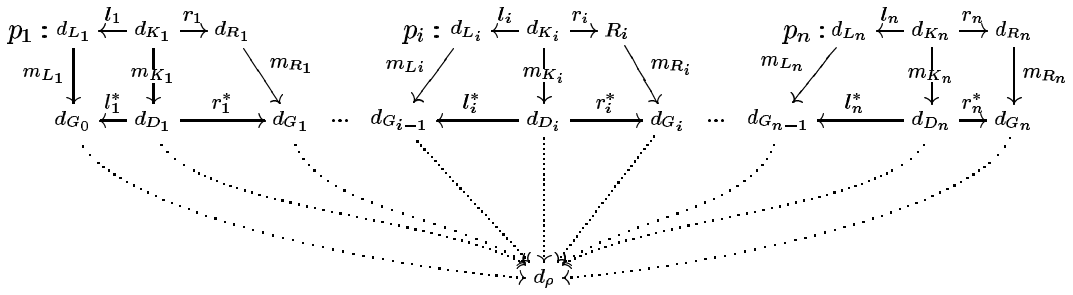

Fig. 11. Colimit construction for derivation $\rho = d_{G_0} \overset{p_1/m_1}{\Longrightarrow} \dots \overset{p_n/m_n}{\Longrightarrow} d_{G_n}$

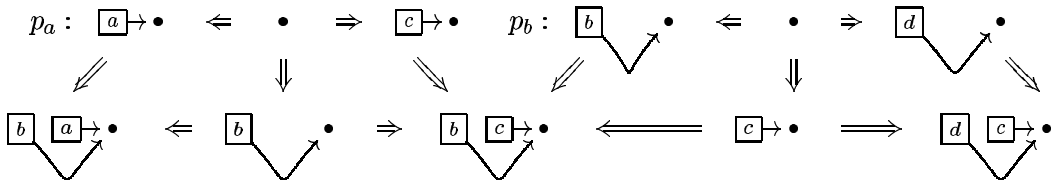

Fig. 12. The derivation $\rho_{ex} = d_{G_0} \overset{p_a/m_a}{\Longrightarrow} d_{G_a} \overset{p_b/m_b}{\Longrightarrow} d_{G_b}$

Definition 5.2 (graph processes) *Let $\mathcal{G}$ be an injective* GTS*, and let ρ be a derivation $d_{G_0} \overset{p_1/m_1}{\Longrightarrow} \dots \overset{p_n/m_n}{\Longrightarrow} d_{G_n}$ of length n (upper part of Figure 11). The (graph) process $\Pi(\rho)$ associated to the derivation ρ is the $n+1$-tuple $\langle t_{G_0}, \langle p_1, \pi_1 \rangle, \dots, \langle p_n, \pi_n \rangle\rangle$: Each π_i is a triple $\langle t_{L_i}, t_{K_i}, t_{R_i} \rangle$, and the graph morphisms $t_{x_i} : d_{x_i} \to d_\rho$, for $x_i \in \{L_i, K_i, R_i\}$ and $i = 1, \dots, n$, are those uniquely induced by the colimit construction shown in Figure 11.*

Let ρ, ρ' be two derivations of length n, both originating from graph d_{G_0}. They are process equivalent *if the associated graph processes are* isomorphic*, i.e., if there exists a graph isomorphism $\gamma_\pi : d_\rho \to d_{\rho'}$ and a bijective function $\gamma_p : \{1, \dots, n\} \to \{1, \dots, n\}$, such that productions p_i and $p'_{\gamma_p(i)}$ coincide for all $i = 1, \dots, n$, and all the involved diagrams commute.* [4]

A graph process associated to a derivation ρ thus includes, by means of the colimit construction and of the morphisms t_{x_i}, the action of each single production p_i on the graph d_ρ. From the image of each d_{x_i} is then possible to recover a suitable partial order among the direct derivations in ρ, which faithfully mirrors the causal relationship among them. For example, let (Σ_{ex}, S_{ex}) be the one-sorted signature containing just four constants, namely $\{a, b, c, d\}$; and let $\mathcal{G}_{ex}$ be the GTS containing two rules, roughly rewriting a into c and b into d. The derivation ρ_{ex} is represented in Figure 12, where, for the sake of readability, graph morphisms are simply depicted as thick arrows.

The process $\Pi(\rho_{ex})$ can be described as in Figure 13, extending the graph $d_{\rho_{ex}}$ with two shaded boxes: They are labelled p_a and p_b, in order to make explicit the mappings t_{x_i} (hence, the action of the rules on the initial graph).

[4] Explicitly, $\gamma_\pi \circ t_{G_0} = t'_{G_0}$, and $\gamma_\pi \circ t_{x_i} = t'_{x_{\gamma_p(i)}}$ for $x_i \in \{L_i, K_i, R_i\}$ and $i = 1, \dots, n$.

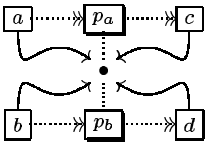

Fig. 13. Compact representation for the process $\Pi(\rho_{ex})$

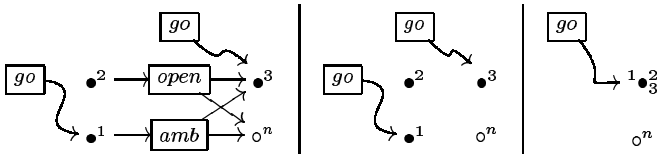

Fig. 14. The rewriting rule for $open\, n.P \mid n[Q] \to P \mid Q$

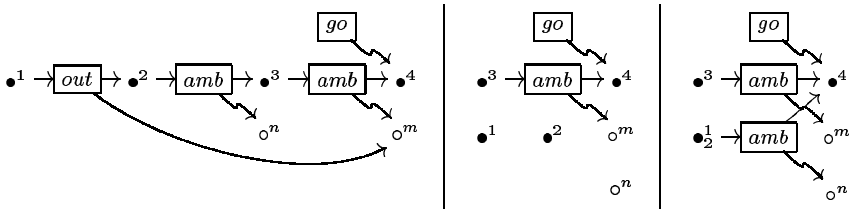

Fig. 15. The rewriting rule for $m[n[out\, m.P \mid Q] \mid R] \to m[R] \mid n[P \mid Q]$

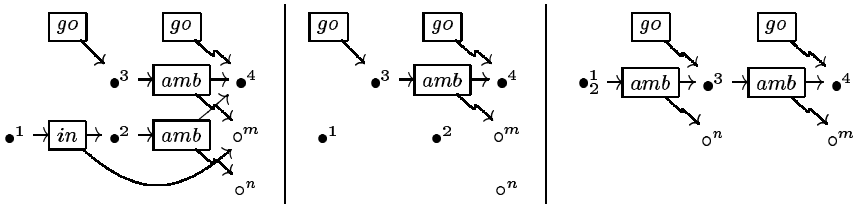

Fig. 16. The rewriting rule for $m[P] \mid n[in\, m.Q \mid R] \to m[n[Q \mid R] \mid P]$

Thus, (the application of) the production p_a consumes the a edge (it is in the image of t_{L_a}, but not in the image of t_{K_a}), and this is denoted by the dotted arrow from a into p_a; it then reads the only node (which is indeed in the image of t_{K_a}), denoted by the dotted arrow with no head; and finally, it creates the c edge, denoted by the dotted arrow into c. Similarly, (the application of) the production p_b consumes the b edge, reads the node and creates the d edge.

We feel confident that our example underlines the connection between the process semantics for graphs, and the standard process semantics for Petri nets. This compact representation is further argued upon on Section 5.3.

5.2 A graph rewriting system for ambients

We finally introduce in this section the graph rewriting system $\mathcal{R}_m$. We first discuss informally its set of productions, then stating more precisely how its rewrites simulate the operational behaviour of processes.

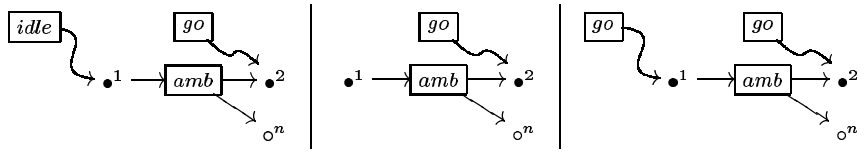

Fig. 17. The rewriting rule for *broadcasting*

The rule $p_{open} : (d_{Lo} \xleftarrow{l_o} d_{Ko} \xrightarrow{r_o} d_{Ro})$ for synchronising an *open* edge with a relevant ambient occurrence is presented in Figure 14: the graph on the left-hand side (center, right-hand side) is d_{Lo} (d_{Ko} and d_{Ro}, respectively); the action of the rule (that is, the span of graph morphisms) is intuitively described by the node identifiers. Both *amb* and *open* edges disappear after reduction, and all the connected nodes are coalesced. Notice that the reduction cannot happen unless both the node shared in the synchronisation and the node under the *amb* prefix are activated, i.e., are linked to an edge labelled by the *go* mark. After reduction, also the node under the *open* prefix becomes activated. The occurrence of the nodes in the interface graph allows for applying the rule in every possible context. Similarly, the occurrence of the *go* operators allows for the simultaneous execution of other derivations using these "tags", since the "read" politics for edges in the interface implies that e.g. more than one pair of distinct resources may synchronise at the top level.

Let us consider now the rules p_{out} and p_{in}, for simulating the *out* and *in* reductions of the calculus, presented in Figure 15 and Figure 16. As for the p_{open} rule, the action of the two productions is described by the node identifiers. It is relevant that the ambients linked with identifier n are first consumed and then re-created by the rules, as they do not belong to the interface graphs. On the contrary, the ambients linked with identifier m are just read, and this implies that e.g. more than one reduction may act simultaneously on that ambient: This fact will be further confirmed when discussing the process semantics for the GTS $\mathcal{R}_m$ in Section 5.3.

Finally, let p_{broad} be the rule in Figure 17. It has no correspondence in the reduction semantics, and its purpose is broadcasting the activation mark to a tree of ambients, whenever its root becomes activated. An occurrence of the *go* operator, denoting an activating point for the process reduction, permeates into the external ambient, reaching the internal node labelled by identifier 1. Of course, the propagation cannot proceed when a capability prefix is reached.

Let the expression $d_G \Longrightarrow_b^* d_H$ denote that d_H is obtained by a finite number of applications of the broadcasting rule p_{broad} to d_G. We can finally state the main theorems of the paper, concerning the soundness and completeness of our encoding with respect to the reduction semantics.

Theorem 5.3 (encoding preserves reductions) *Let P, Q be processes, and let Γ be a set of names such that $fn(P) \subseteq \Gamma$. If the reduction $P \rightarrow_f Q$ is entailed, then $\mathcal{R}_m$ entails a derivation $\{\!|P|\!\}_\Gamma \Longrightarrow_b^* d_G \Longrightarrow d_H$, such that $\{\!|Q|\!\}_\Gamma \Longrightarrow_b^* d_H$.*

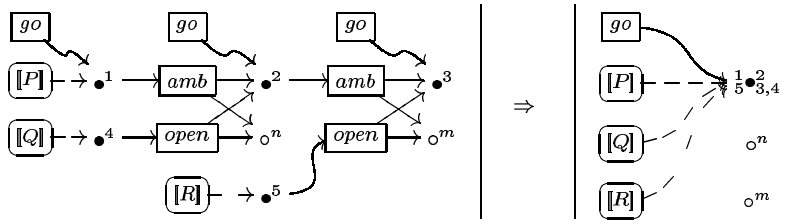

Fig. 18. Simultaneous application of nested, yet causally unrelated reductions

Intuitively, process reduction is simulated by first applying a sequence of broadcasting rules, thus enabling (by the propagation of the *go* operator) those events whose activating point is nested inside one or more ambients, and then simulating the actual reduction step. The mapping $\{\!|P|\!\}_\Gamma$ introduced in the statement of the theorem denotes the graph (that is, a representative of the equivalence class of isomorphic graphs) underlying the term graph $[\![P]\!]^{go}_\Gamma$.

Theorem 5.4 (encoding does not add reductions) *Let P be a process, and let Γ be a set of names such that $fn(P) \subseteq \Gamma$. If $\mathcal{R}_m$ entails a derivation $\{\!|P|\!\}_\Gamma \Longrightarrow^*_b d_G \Longrightarrow d_H$, then there exists a process Q such that $P \rightarrow_f Q$ is entailed and $\{\!|Q|\!\}_\Gamma \Longrightarrow^*_b d_H$.*

5.3 On causal dependency and simultaneous execution

We argued in the Introduction that the concurrent semantics of GTS's may shed some light in the understanding of process behaviour for mobile ambients.

It is in fact an obvious consideration that by our encoding we can equip mobile ambients with a concurrent semantics, simply considering for each process P of the calculus the classes of process equivalent derivations associated to the graph $\{\!|P|\!\}_{fn(P)}$. This is intuitively confirmed by the analysis of a rather simple process, namely, $S = m[n[P] \mid open\, n.Q] \mid open\, m.R$. The process S may obviously perform two reductions, opening either the ambient m, or the ambient n: These reductions should be considered as independent, since they act on nested, yet causally unrelated occurrences of an ambient. This independence becomes explicit in the graph d_S, obtained by applying twice the broadcasting rule to $\{\!|S|\!\}_{\{m,n\}}$, and depicted on the left-hand-side of Figure 18 (forgetting for the sake of clarity the subscripts and the dashed arrows leaving from the graphs underlying $[\![P]\!]_{\{m,n\}}$ and $[\![Q]\!]_{\{m,n\}}$ and directed to either m or n). Production p_{open} may now be applied twice, reducing either those edges linked with the node n, or those linked with the node m, thus simulating the reductions originating from S. These rewrites may be executed in any order, resulting in two different derivations, which are nevertheless process equivalent. The resulting graph is depicted on the right-hand side of Figure 18.

Let us consider now a more complex example, and let T be the process $m[n[out\, m.P] \mid R \mid o[out\, m.Q]]$, which can be reduced into $n[P] \mid m[R] \mid o[Q]$ by applying twice the *out* reduction on ambient m, and depicted in Figure 19.

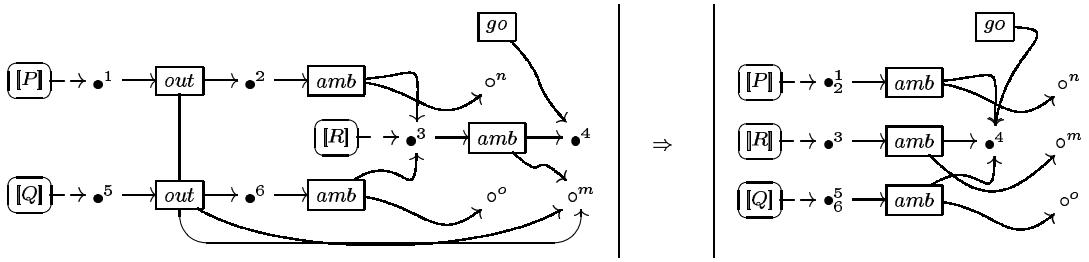

Fig. 19. Simultaneous application of nested reductions sharing an ambient

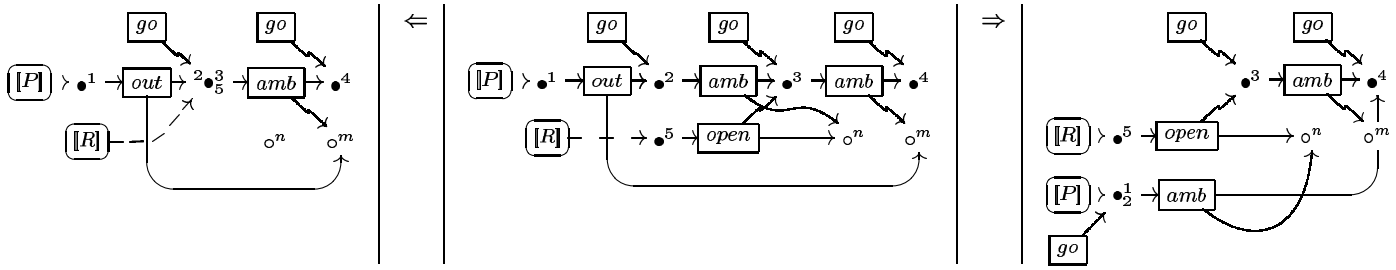

Fig. 20. Grave interference as symmetric conflict

The two rules may be applied simultaneously, since the occurrence of the *amb* operator, linked to the node with identifier m, is shared. The process resulting from the colimit construction of Figure 11, if represented as in Figure 13, contains two events: The first one consumes the *out* edge linked with nodes 1, 2 and m, and the *amb* edge linked with nodes 2, 3 and n; reads the *amb* edge linked with nodes 3, 4 and m (and all the related nodes); and creates the *amb* edge linked with nodes $1 = 2$, 4 and n. Symmetrically, the other consumes the *out* edge linked with nodes 5, 6 and m, and the *amb* edge linked with nodes 6, 3 and o; reads the *amb* edge linked with nodes 3, 4 and m (and all the related nodes); and creates the *amb* edge linked with nodes $5 = 6$, 4 and o.

Let U be the process $m[n[out\, m.P] \mid open\, n.R]$. This is listed by Levi and Sangiorgi [22] as an example of *grave interference*, representing a situation in the calculus that should be deprecated, and actually "should be regarded as a programming error". The execution of the internal *out* reduction on the ambient m destroys the possibility to perform the execution of the external *open* reduction on the ambient n, and vice versa. This is confirmed by the analysis of the graph in the middle of Figure 20, obtained by applying twice the broadcasting rule to $\{\!|U|\!\}_\Gamma$. The two derivations originating from that graph, and simulating the execution of the two reductions, are represented on the right-hand-side (the internal *out*) and on the left-hand-side (the external *open*). These derivations can not be extended with additional steps, in order to become process equivalent. This situation is usually described by saying that the two derivations denote a *symmetric conflict* of events.

More interestingly, let us consider an apparently similar instance of grave interference, represented by the process $V = m[n[out\, m.P] \mid Q] \mid open\, m.R$.

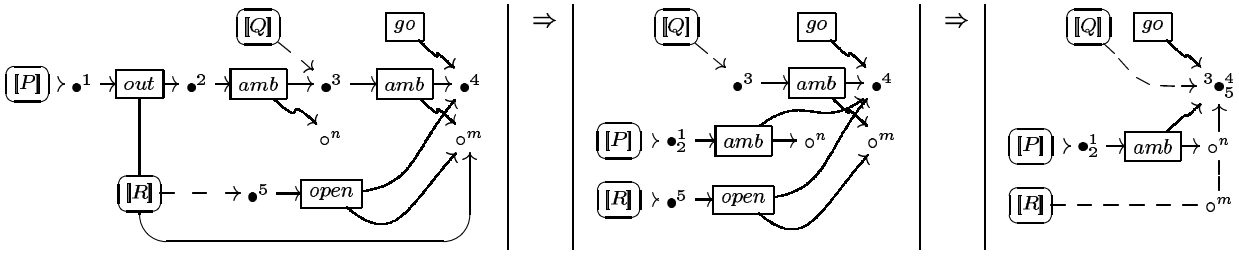

Fig. 21. Grave interference as asymmetric conflict

The external *open* reduction on ambient m destroys the possibility to perform the internal *out* reduction on the same ambient, but *the vice versa does not hold*. After the execution of the internal *out* reduction, an external *open* may be performed, and the two applications of p_{open} represent *the same event*. Since the occurrence of the *amb* operator is only read by p_{out} of Figure 15, the same operator is available after the rewriting step. We are thus facing an *asymmetric conflict*, lifting the notion from a recent extension of the event structures formalism [2]. The graph $\{\!|V|\!\}_{\{m,n\}}$ is represented on the left-hand side of Figure 21; the graphs obtained by first the application of p_{out}, and then of p_{open}, are represented on the center and on the right-hand side of the figure.

6 Conclusions and Further Works

We presented an encoding for finite, communication-free processes of the mobile ambients calculus into term graphs, proving its soundness and completeness with respect to the original, interleaving operational semantics.

With respect to most of the other approaches for the graphical implementation of calculi with name mobility (see e.g. Milner's π-*nets* [23], Parrow 's *interaction diagrams* [26], Gardner's *process frameworks* [14], Hasegawa's *sharing graphs* [16], Montanari and Pistore's presentation of π-calculus by DPO rules [25] or König SPIDER *calculus* [21]; an exception are Yoshida's *concurrent combinators* [27]), we considered unstructured (that is, non hierarchical) graphs, thus avoiding any "encapsulation" of processes. The implication is twofold. First of all, it allows the reuse of standard graph rewriting theory and tools for simulating the reduction semantics, such as e.g. the DPO formalism and the HOPS programming system [20]. More importantly, it allows for the simultaneous execution of independent reductions, which are nested inside ambients, and possibly share some resource. While this feature is less relevant for e.g. the π-calculus, where each process can be considered just a soup of disjoint sequential agents (much in the spirit of Berry's and Boudol's CHAM approach [4]), it is relevant in the present context, where ambients are nested, and yet can be "permeated" by a reduction. A first, rough analysis is performed in Section 5.3, and we plan to extend our preliminary considerations to a non-deterministic concurrent semantics for mobile ambients, much in the spirit of the event structure semantics developed in [1].

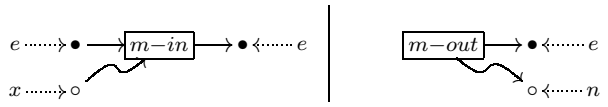

Fig. 22. Term graphs for input (x) and asynchronous output $\langle n \rangle$ actions.

Our encoding can be extended to recover the communication primitives, as long as we restrict communication to name passing: The graphs for encoding input and asynchronous output actions are depicted in Figure 22. In fact, we feel confident that any calculus with name mobility may find a presentation within our formalism, along the line of the encoding for mobile ambients. The calculus should of course contain a parallel operator which is associative, commutative and with an identity; moreover, its operational semantics should be reduction-like (i.e., expressed by unlabelled transitions), and the rules should never substitute a free name for another, so that name substitution can be handled by node coalescing (with a mechanism reminiscent of *name fusion*).

It should be noted that any monoidal category with a suitable enrichment (namely, where each object a is equipped with two monoidal transformations $a \times a \to a$ and $1 \to a$, making it a *monoid*) could be used as a sound model for the encoding. The relevant thing is that, among this class of models, (a suitable sub-category of) the category $\mathbf{RG}_{\Sigma,S}$ of graphs as objects, and ranked graphs as morphisms, is the initial one [5,7], so that Proposition 4.3 is just a corollary of this general result. Our work is thus tightly linked with ongoing research on the graphical presentations for categorical formalisms, as e.g. on *premonoidal* [17] and *traced monoidal* [19] categories. More importantly, also graph processes may be equipped with an algebraic structure [8,12], thus providing a formalism for denoting also reductions in mobile ambients.

As for the finiteness conditions, it is a different matter. In fact, it is a difficult task to recover the behaviour of processes including a *replication* operator, since replication is a global operation, involving the duplication of necessarily unspecified sub-processes, and it is hence hard to model *via* graph rewriting, which is an eminently local process. Nevertheless, our framework allows for the modeling of *recursive* processes, that is, defined using constant invocation, so that a process is a family of judgments of the kind $A = P$. Thus, each process is compiled into a different graph transformation system, adding to the four basic rewriting rules a new production p_A for each constant A, intuitively simulating the unfolding step $\{\!|A|\!\}_\Gamma \Rightarrow \{\!|P|\!\}_\Gamma$, for a suitable Γ.

References

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

[2] P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Foundations of Software Science and Computation Structures*, volume 1378 of *Lect. Notes in Comp. Science*, pages 63–80. Springer, 1998. Revised version to appear in *Information and Computation*.

[3] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph reduction. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Parallel Architectures and Languages Europe*, volume 259 of *Lect. Notes in Comp. Science*, pages 141–158. Springer, 1987.

[4] G. Berry and G. Boudol. The chemical abstract machine. *Theoret. Comput. Sci.*, 96:217–248, 1992.

[5] R. Bruni, F. Gadducci, and U. Montanari. Normal forms for algebras of connections. *Theoret. Comput. Sci.*, 2001. To appear. Available at `http://www.di.unipi.it/~ugo/tiles.html`.

[6] L. Cardelli and A. Gordon. Mobile ambients. In M. Nivat, editor, *Foundations of Software Science and Computation Structures*, volume 1378 of *Lect. Notes in Comp. Science*, pages 140–155. Springer, 1998.

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

[8] A. Corradini and F. Gadducci. Rewriting on cyclic structures: Equivalence between the operational and the categorical description. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 33:467–493, 1999.

[9] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.

[10] F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.

[11] G. Ferrari and U. Montanari. Towards the unification of models for concurrency. In A. Arnold, editor, *Trees in Algebra and Programming*, volume 431 of *Lect. Notes in Comp. Science*, pages 162–176. Springer, 1990.

[12] F. Gadducci, R. Heckel, and M. Llabrés. A bi-categorical axiomatisation of concurrent graph rewriting. In M. Hofmann, D. Pavlovič, and G. Rosolini, editors, *Category Theory and Computer Science*, volume 29 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1999. Available at `http://www.elsevier.nl/locate/entcs/volume29.html/`.

[13] F. Gadducci and U. Montanari. Comparing logics for rewriting: Rewriting logic, action calculi and tile logic. *Theoret. Comput. Sci.*, 2001. To appear. Available at `http://www.di.unipi.it/~ugo/tiles.html`.

[14] Ph. Gardner. From process calculi to process frameworks. In C. Palamidessi, editor, *Concurrency Theory*, volume 1877 of *Lect. Notes in Comp. Science*, pages 69–88. Springer, 2000.

[15] U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.

[16] M. Hasegawa. *Models of Sharing Graphs*. PhD thesis, University of Edinburgh, Department of Computer Science, 1997.

[17] A. Jeffrey. Premonoidal categories and a graphical view of programs. Technical report, School of Cognitive and Computing Sciences, University of Sussex, 1997. Available at `http://www.cogs.susx.ac.uk/users/alanje/premon/`.

[18] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.

[19] A. Joyal, R.H. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:425–446, 1996.

[20] W. Kahl. The term graph programming system HOPS. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, pages 136–149. Springer, 1999. Tool available at `http://ist.unibw-muenchen.de/kahl/HOPS/`.

[21] B. König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.

[22] F. Levi and D. Sangiorgi. Controlling interference in ambients. In T. Reps, editor, *Principles of Programming Languages*, pages 352–364. ACM Press, 2000.

[23] R. Milner. Pi-nets: A graphical formalism. In D. Sannella, editor, *European Symposium on Programming*, volume 788 of *Lect. Notes in Comp. Science*, pages 26–42. Springer, 1995.

[24] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. Part I and II. *Information and Computation*, 100:1–77, 1992.

[25] U. Montanari and M. Pistore. Concurrent semantics for the π-calculus. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Computer Science*. Elsevier Science, 1995.

[26] J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, 2:407–443, 1995.

[27] N. Yoshida. Graph notation for concurrent combinators. In T. Ito and A. Yonezawa, editors, *Theory and Practice of Parallel Programming*, volume 907 of *Lect. Notes in Comp. Science*, pages 393–412. Springer, 1994.