Modelling Calculi with Name Mobility using Graphs with Equivalences

Paolo Baldan[1]

Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

Fabio Gadducci[2] and Ugo Montanari[3]

Dipartimento di Informatica, Università di Pisa, Italy

Abstract

In the theory of graph rewriting, the use of coalescing rules, i.e., of rules which besides deleting and generating graph items, can coalesce some parts of the graph, turns out to be quite useful for modelling purposes, but, at the same time, problematic for the development of a satisfactory partial order concurrent semantics for rewrites. Rewriting over graphs with equivalences, i.e., (typed hyper)-graphs equipped with an equivalence over nodes provides a technically convenient replacement of graph rewriting with coalescing rules, for which a truly concurrent semantics can be easily defined. The expressivity of such a formalism is tested in a setting where coalescing rules typically play a basic role: the encoding of calculi with name passing as graph rewriting systems. Specifically, we show how the (monadic fragment) of the solo calculus, one of the dialect of those calculi whose distinctive feature is name fusion, can be encoded as a rewriting system over graph with equivalences.

Keywords: Concurrent graph rewriting, DPO approach, graphical encoding of nominal calculi, graph process semantics.

1 Introduction

Recent years have seen an increasing use of graphical formalisms for the modelling of concurrent and distributed systems. Graph-like structures naturally provide a formal yet flexible view of system states, while the rewriting rules suitably model local state transformations. Among the different formalisms proposed in the literature, the so-called *double pushout* (DPO) approach offers a large variety of theoretical and practical tools for the visual specification of a system (see, e.g., [16,7]), abstracting away from the unnecessary details of the state representation. As an example,

⋆ Research partially supported the EC RTN 2-2001-00346 SegraVis, the EU IST-FP6-16004 SEnSOria and the MIUR project ART.

[1] Email: baldan@dsi.unive.it

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

[3] Email: ugo@di.unipi.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the DPO rewriting techniques for simulating reductions in nominal calculi [14,2], as presented in [8,9], view a (possibly recursive) process as a graph, thus modelling reductions by rewrites. The use of graphs gets rid of the problems concerning the implementation of reduction over the structural congruence, such as α-conversion of (bound) names, since equivalent processes are mapped into isomorphic graphs.

However, the diffusion of the formalism raises unresolved issues concerning the analysis of its concurrency aspects. Consider again the graphical encodings for nominal calculi mentioned above: a concurrent semantics for the graph rewriting formalism would provide a concurrent semantics for process reduction, but unfortunately these encodings fall outside the canon of DPO concurrent semantics. More specifically, the matching morphisms (those morphisms identifying the occurrence of the left-hand side of a rule into the graph to be rewritten) are forced to be injective. More importantly, the right-hand side of the rules resulting from the encoding are specified by non-injective morphisms (operationally, they force some node coalescing in the graph to be rewritten). On the one hand, such features prevents the development of a satisfactory partial order concurrent semantics of rewrites; on the other hand, they are general enough to deserve to be addressed.

In order to allow the use of coalescing rules, while retaining a satisfactory theory of concurrency, we advocate the use of rewriting over a family of structures, called *graphs with equivalences*: ordinary (hyper-)graphs equipped with an equivalence relation over their nodes. The underlying intuition is simple: the coalescing of nodes is replaced by the handling of equivalence classes over nodes. Avoiding the fusion of these graph items (and thus preserving the identities of the nodes involved in a computation) allows to recover the theoretical results associated to the concurrent features of the DPO approach.

Here we present the basics of rewriting over graphs with equivalences and we illustrate the main ideas by testing the formalism against the encoding of the monadic *solo calculus* [13], one of the dialects of those nominal calculi whose distinctive feature is name fusion [11,15]. This is intended to suggest that the formalism of graphs with equivalences is expressive enough to properly recast the graphical encodings of nominal calculi proposed in e.g. [8,9]. The presentation is mainly of informal nature. A more detailed presentation, including the development of the concurrency theory for the formalism can be found in [1] (where, however, only the specification of a fragment of the solo calculus was presented).

The paper has the following structure. Section 2 describes the formalism of graphs with equivalences, which is amenable to the DPO approach to rewriting. Section 3 presents an encoding of the processes of the solo calculus into graphs with equivalences, which guarantees a correspondence between process reductions and graph derivations. Finally, Section 4 concludes the paper, discussing open issues and directions of future research.

2 Rewriting Graphs with Equivalences

This section introduces graphs with equivalences, i.e., graphs endowed with an equivalence over the set of nodes. Rewriting systems over such structures are proposed as a replacement of rewriting over ordinary graphs with node coalescing rules.

2.1 The Category of Graphs with Equivalences

A *(hyper-)graph* G is a tuple $\langle V_G, E_G, c_G\rangle$ for V_G the set of nodes, E_G the set of edges and $c_G : E_G \to V_G^*$ the connection function. A (hyper-)graph morphism $f : G \to H$ is a pair $f = \langle f_V : V_G \to V_H, f_E : E_G \to E_H\rangle$ satisfying $c_H(f_E(e)) = f_V^*(c_G(e))$ for any $e \in E_G$. The corresponding category is denoted by **Graph**.

Definition 2.1 (graphs with equivalences) *A* graph with equivalences (e-graph) *is a pair* $\mathbb{G} = \langle G, \sim_G\rangle$ *where* G *is a graph and* $\sim_G \subseteq V_G \times V_G$ *is an equivalence over the set of nodes. Given two e-graphs* $\mathbb{G}$ *and* $\mathbb{H}$*, a morphism* $f : \mathbb{G} \to \mathbb{H}$ *is a graph morphism* $f : G \to H$ *such that for all* $n, n' \in V_G$*, if* $n \sim_G n'$ *then* $f(n) \sim_H f(n')$*. The category of e-graphs and their morphisms is denoted by* **EGraph**.

An e-graph $\mathbb{G}$ provides an alternative representation for the graph $G/_{\sim_G}$ obtained by quotienting G with respect to $\sim_G$. Formally, we can define a quotient functor $\mathcal{Q} : \mathbf{EGraph} \to \mathbf{Graph}$ defined on objects as $\mathcal{Q}(\mathbb{G}) = G/_{\sim_G} = \langle V/_{\sim_G}, E, c'\rangle$ where $c'([e]_{\sim_G}) = [v_1]_{\sim_G} \dots [v_n]_{\sim_G}$ if $c(e) = v_1 \dots v_n$. Given $f : \mathbb{G} \to \mathbb{H}$ we have $\mathcal{Q}(f)$ defined by $\mathcal{Q}(f)([v]_{\sim_G}) = [f(v)]_{\sim_G}$. An example of graph with equivalences can be found in Fig. 5(a). Equivalence classes are represented by a dotted rectangle, encompassing those nodes belonging to the class. In the example there are four equivalence classes: $\{y_1, y_2\}$, $\{x_1, x_2\}$, $\{z_1, z_2\}$ and $\{w, w_1, w_2\}$. The corresponding quotient graph, obtained by collapsing equivalent nodes, is depicted in Fig. 5(b).

In order to define rewriting over e-graphs some considerations are in order.

Observe that monos in **EGraph** are morphisms $f : \mathbb{G} \to \mathbb{H}$ such that $f : G \to H$ is a mono in **Graph**. *Regular* monos are monos $f : \mathbb{G} \to \mathbb{H}$ which reflect as well as preserve the equivalences of nodes, i.e., such that for all $n, n' \in V_G$ if $f(n) \sim_H f(n')$ then $n \sim_G n'$. Note that regular monos over e-graphs induce monos over the corresponding quotient graphs, i.e., if $f : \mathbb{G} \to \mathbb{H}$ is regular mono then $\mathcal{Q}(f) : \mathcal{Q}(\mathbb{G}) \to \mathcal{Q}(\mathbb{H})$ is injective.

The category **EGraph** has all pushouts, which are computed by taking the pushout in **Graph**, endowed with the equivalence arising as the "union" of the equivalences of the components.

2.2 Rewriting e-graphs

We next define rewriting systems over e-graphs according to the algebraic double-pushout (DPO) approach to rewriting, as presented in [4,5]. For technical reasons it is convenient to work with *typed e-graphs*, which are e-graphs labelled over a structure that is itself an e-graph (see, e.g., [3] for the idea of graph typing).

Given an e-graph $\mathbb{T}$, the category of e-graphs typed over $\mathbb{T}$ is the slice category **EGraph** $\downarrow \mathbb{T}$, later denoted $\mathbb{T}$-**EGraph**. Explicitly, the objects of the category are the e-graph morphisms $f : \mathbb{G} \to \mathbb{T}$ with target $\mathbb{T}$, and arrows are e-graph morphisms making the obvious diagram commutes. Given a $\mathbb{T}$-typed e-graph $\mathbb{G}$, we write $|\mathbb{G}|$ for the underlying e-graph and $t_{\mathbb{G}}$ for the typing arrow $t_{\mathbb{G}} : |\mathbb{G}| \to \mathbb{T}$.

Rewriting systems over typed e-graphs will be used as a replacement of rewriting systems over ordinary graphs where rules can coalesce nodes. Intuitively, the coalescing of nodes in rewriting systems over graphs becomes the generation of an equivalence between such nodes in the setting of e-graphs.

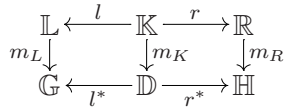

Fig. 1. A direct derivation.

Definition 2.2 (e-graph production) *A $\mathbb{T}$-typed* e-graph production *is a span $\mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$ in $\mathbb{T}$-***EGraph** *such that l and r are mono. It is called left-linear if l is regular mono. A* typed e-graph transformation system *(*e-GTS*) is a tuple $\langle \mathbb{T}, P, \pi \rangle$ where $\mathbb{T}$ is a fixed graph, P is a set of production names, and π is a function mapping each name to a $\mathbb{T}$-typed production. An e-*GTS *is called* left-linear *if all its productions are left-linear.*

Given a left-linear production p, in the production $\mathcal{Q}(\mathbb{L}) \xleftarrow{\mathcal{Q}(l)} \mathcal{Q}(\mathbb{K}) \xrightarrow{\mathcal{Q}(r)} \mathcal{Q}(\mathbb{R})$ the left morphism is mono, while the right morphism may coalesce nodes.

An example of left-linear e-graph production can be found in Fig. 7. Note that the right morphism is mono, but not regular mono since node equivalence is not reflected: the equivalence classes $\{y\}$ and $\{w_1, w_2\}$ in the interface are "merged" in a single class $\{y, w_1, w_2\}$ in the right-hand side e-graph.

Definition 2.3 (derivation) *Given a $\mathbb{T}$-typed production $p : \mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$, a match of p in a $\mathbb{T}$-typed e-graph $\mathbb{G}$ is a morphism $m_L : \mathbb{L} \to \mathbb{G}$. A* direct derivation *from $\mathbb{G}$ to $\mathbb{H}$ via production p at a match m is a diagram as depicted in Fig. 1, where (1) and (2) are pushout squares in $\mathbb{T}$-***EGraph***. It is called* strict *if the match is regular mono. We write $\mathbb{G} \overset{p/m}{\Longrightarrow} \mathbb{H}$, where $m = \langle m_L, m_K, m_R \rangle$, or simply $\mathbb{G} \Longrightarrow \mathbb{H}$.*

Roughly, concerning the graphical part, the application of a production p first removes all the items of G matched by $L - l(K)$, leading to the context graph D. Then the items of $R - r(K)$ are added to D, thus obtaining H.

Concerning the equivalence part, being l a regular mono intuitively means that equivalences among nodes are never deleted, that is, two nodes which are equivalent in the e-graph $\mathbb{L}$ will still be equivalent in the e-graph $\mathbb{R}$. Hence, the equivalence in $\mathbb{D}$ is just the restriction of the equivalence in $\mathbb{G}$. Instead, whenever r is not regular mono, as an effect of taking the second pushout, some nodes which were not equivalent in $\mathbb{D}$ might become equivalent in $\mathbb{H}$. On the formal side, the regular mono requirement for l ensures that the pushout complement, when it exists, is unique.

In several applications it is necessary to consider injective matches only. For e-graphs, this property corresponds to the requirement of having regular mono matches. For this reason, the rest of the paper will focus on strict derivations and left-linear e-GTS, hence both qualifications "strict" and "left-linear" will be omitted.

A drawback of the approach is given by the fact that a single node of a "standard" graph can be represented in an e-graph by an equivalence class of possibly unbounded size. Therefore, in order to model node deletion, an unbounded number of rules deleting equivalence classes of arbitrary size should be inserted into a transformation system. However, note that for modelling purposes, it is often not restrictive to consider only rules which never delete nodes: indeed, this happens on most graphical encodings of process calculi. Node deletion is then simulated by leaving a node isolated, thus assuming a mechanism for performing garbage collection.

$$P \mid Q = Q \mid P \qquad P \mid \mathsf{0} = P \qquad P \mid (Q \mid R) = (P \mid Q) \mid R$$
$$(\nu x)[y = w]P = [y = w](\nu x)P \quad \text{for } x \notin \{y, w\}$$
$$(\nu x)(\nu y)P = (\nu y)(\nu x)P \qquad (\nu x)(P \mid Q) = P \mid (\nu x)Q \quad \text{for } x \notin \mathit{fn}(P)$$

Fig. 2. Structural axioms for the solo calculus.

3 Encoding a simple process calculus

In this section we put the e-graph formalism at work, showing that it allows for encoding a simple (the simplest available, in fact) process calculus, namely, the monadic *solo calculus* [13], one of the dialects of those nominal calculi whose distinctive feature is name fusion [11,15].

3.1 The monadic fragment of the solo calculus

We shortly introduce the monadic variant of the *solo calculus*, its structural equivalence and the associated reduction semantics.

Definition 3.1 (processes) *Let $\mathcal{N}$ be a set of* names*, ranged over by $x, y, w, \ldots$. The set of* processes Proc *is generated by the syntax*

$$P ::= \mathsf{0},\ \sigma,\ (\nu x)P,\ P_1 \mid P_2,\ [x = y]P \qquad \textit{for } \sigma \in \{x(y), \overline{x}y\}$$

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

The operators $x(y)$ and $\overline{x}y$ are denoted as *input* and *output*, respectively, even if their symmetric behaviour makes the distinction (typical of most calculi) immaterial; collectively, each instance of them is called a *solo*, to emphasise its lack of connections, except for name sharing, with the other components. Finally, the first argument of the two operators, indicated by x, is usually called the *channel* where the communication of information takes place.

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 $(\nu y)P$: the name y is bound in P, and it can be freely α-converted. Using these definitions, the behaviour of a process P is described as a relation obtained by closing a set of basic rules under a suitable congruence.

Definition 3.2 (reduction semantics) *The* reduction relation *for processes is the relation $R_\sigma \subseteq Proc \times Proc$, closed under the structural congruence $\equiv$ induced by the set of equations in Fig. 2, generated by the set of inference rules in Fig. 3, where $P \to Q$ means that $\langle P, Q\rangle \in R_\sigma$.*

Rules (r_1) and (r_2) characterise the communication between restricted processes. Consider, for instance, rule (r_2): the process $\overline{x}w$ is ready to communicate the (possibly global) name w along the channel x; it then synchronises with the process $x(y)$, and the bound name y is thus substituted by w on *all the occurrences* inside the residual process P. Hence, communication has a global effect affecting the process as a whole. Rule (r_3) expresses the fact that there is no reason to bind a name during a reduction, if no substitution has actually to occur. Rule (r_4) removes those matches that hold true. Finally, rules (r_5) and (r_6) simply state the closure of the reduction relation with respect to the operators of restriction and parallel composition.

$$\frac{y \neq w}{(\nu w)(x(y) \mid \overline{x}w \mid P) \to P\{^{y}/_{w}\}}\ (r_1) \qquad \frac{}{[x = x]P \to P}\ (r_4)$$

$$\frac{y \neq w}{(\nu y)(x(y) \mid \overline{x}w \mid P) \to P\{^{w}/_{y}\}}\ (r_2) \qquad \frac{P \to Q}{(\nu x)P \to (\nu x)Q}\ (r_5)$$

$$\frac{}{x(y) \mid \overline{x}y \to \mathtt{0}}\ (r_3) \qquad \frac{P \to Q}{P \mid R \to Q \mid R}\ (r_6)$$

Fig. 3. Inference rules for the solo calculus.

There are two differences with respect to the operational semantics for the monadic fragment of the calculus proposed in [13]. The first difference, of a syntactical nature, is the explicit presentation of the four reduction rules, which in [13] are summarised as a unique rule equipped with constraints on the substitution induced by the name fusion. The second difference is the lack of two axioms, namely, $[x = x]P = P$ and $(\nu x)\mathtt{0} = \mathtt{0}$, which are responsible for the garbage collection of useless operators. However, the calculus is essentially the same. In fact, let $=_e$ denote the equivalence on processes induced by extending the structural axioms of Fig. 2 with the previously mentioned laws for match and restriction, and let $\to_e$ be the induced reduction relation. Then, if $P \to_e Q$, there exists a process R such that $P \to R$ and $R =_e Q$.

We conclude with a remark on the expressiveness of the calculus. Despite their simple syntax and operational semantics, both the monadic variant and the dyadic variant without match (solos come with two names, besides the channel) are as expressive as the full fusion calculus (as proved in [13]), which in turn is a symmetric version of the foremost nominal calculi, the π-calculus [14].

3.2 The graphical encoding of solos

This section presents an encoding of the solo calculus based on e-graphs. It resembles the encoding using standard graphs presented in [10, Section 5], but here node coalescing rules are replaced with rules generating node equivalences. We do not provide a formal definition of the encoding, which, however, can be easily obtained by adapting the proposals for mobile ambients and π-calculus in [8,9].

In order to help intuition, we begin with a description of a suitable normal form for structurally congruent processes. Let us say that a process is *basic* if it contains no restriction operator; and it is *disjoint* if it is basic and all the names occurring in its operators are different.

Lemma 3.3 (normal form) *Any process P is equivalent to a process of the shape $(\nu x_1) \dots (\nu x_n)B$ where all x_i's are different and B is a basic process.*

A process in normal form is denoted as $(\nu X)B$, where $X = \{x_1, \dots, x_n\}$ is a set of names, since the order of the restriction operators is immaterial.

Definition 3.4 (disjoint normal form) *Let P be a process and let $(\nu X)B$ be its normal form. We call* disjoint normal form *of P an expression of the kind $(\nu X)D\xi$, where D is a disjoint process, such that $X \cap \mathit{fn}(D) = \emptyset$ and $\xi : \mathit{fn}(D) \to \mathit{fn}(B)$ is a surjective name substitution satisfying $D\xi = B$.*

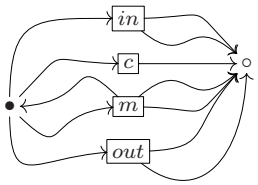

Fig. 4. The type graph T_σ.

After renaming the basic process, the substitution ξ picks a representative for each equivalence class of names. For example, the process $P_e = (\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{y}z)$ is described by the disjoint normal form $(\nu w)D_e\xi_e$ where $D_e = x_2(y_2) \mid \overline{x}_1 w_2 \mid w_1(z_2) \mid \overline{y}_1 z_1$ and ξ_e is the obvious substitution.

The above characterisation naturally suggests a representation for processes based on typed e-graphs. For technical reasons, the graph underlying the e-graph representation of a process will be completely disconnected, i.e., all edges will be incident to distinct nodes.

The type e-graph $\mathbb{T}_\sigma$ is represented in Fig. 4. It includes two nodes: intuitively the black node correspond to the sort of processes, while the white node correspond to the sort of names. Nodes typed over such items will be called process nodes and name nodes, respectively. Additionally, $\mathbb{T}_\sigma$ includes four different edges, corresponding to the operators of the calculus. The equivalence on nodes is the identity, i.e., $\mathbb{T}_\sigma$ is essentially a standard graph. The typing for nodes will be represented by using the shape of the nodes themselves, while for edges we will use labels *in*, *out*, *c* and *m*.

Let P be a process, and $(\nu X)D\xi$ its disjoint normal form. Then, in the typed e-graph $\mathbb{G}_P$ associated to P any name occurring in D becomes a name node. Moreover solos and all operators, except parallel, become edges. Note that in this simple calculus almost all the edges are connected to a single process node (intuitively the process they belong to). The only exception is the match operator which has a continuation and thus the corresponding edge is connected to two process nodes. Since all the edges must be disconnected from each other, $\mathbb{G}_P$ has as many process nodes as the occurrences of solos and restriction operators, plus twice those of the match operators. The effect of the substitution ξ is represented by the equivalence $\sim_{G_P}$ between name nodes: given two nodes x and y we have $x \sim_{G_P} y$ iff $\xi(x) = \xi(y)$ or $\xi(x) = y$. Similarly, the topological structure is represented by the equivalence on process nodes.

Consider again the process $P_e = (\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{y}z)$ and its disjoint normal form. Its encoding is represented in Fig. 5(a), where name nodes, for the sake of clarity, are additionally equipped with the name they represent, in order to make the correspondence between the disjoint normal form and the encoding clearer. Some intuition may be gained by looking at the quotient graph $\mathcal{Q}(\mathbb{G}_{P_e})$ depicted in Fig. 5(b), obtained by collapsing equivalent nodes: it directly corresponds to the basic process P_e, which is already in normal form, and this was indeed the encoding proposed for process P_e in [10, Fig. 11].

The encoding of process $P_m = (\nu w)([y = w].\mathtt{0} \mid x(y) \mid \bar{x}w)$ can be found in Fig. 6. A disjoint normal form, in this case, can be $P_m = (\nu w)D_m\xi$, where $D_m = [y_2 = w_2].\mathtt{0} \mid x_1(y_1) \mid \bar{x}_2 w_1$ and ξ is the obvious substitution.

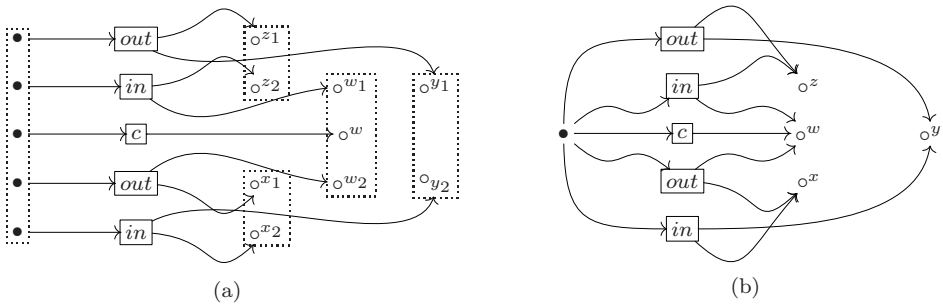

Fig. 5. (a) The e-graph encoding of process P_e and (b) the quotient graph $\mathcal{Q}(P_e)$.

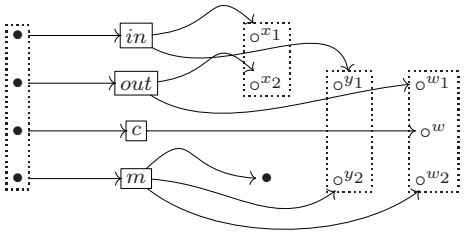

Fig. 6. The e-graph encoding of the process P_m.

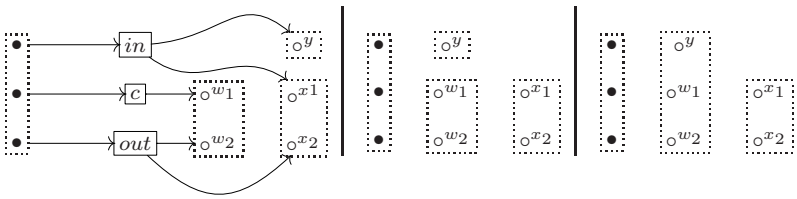

Fig. 7. A production of $\mathcal{G}_\sigma$ encoding reduction (r_1).

3.3 Encoding the rules

We now introduce the e-GTS $\mathcal{G}_\sigma$ in $\mathbb{T}_\sigma$-**EGraph**. It basically contains just four productions (i.e., one for each rule of the reduction system), plus some "instances" of them. The first production p_1^σ is depicted in Fig. 7: the e-graph on the left-hand side (center, right-hand side) is $\mathbb{L}_1^\sigma$ ($\mathbb{K}_1^\sigma$ and $\mathbb{R}_1^\sigma$, respectively). The effect of the rule is described by the names of the nodes: as an example, the nodes identified by y and w_i's, which are distinct in $\mathbb{L}_1^\sigma$, are made equivalent in $\mathbb{R}_1^\sigma$. Please note that the node identifiers are of course arbitrary: they correspond to the actual elements of the set of nodes, and they are used just to represent the span of morphisms.

The rule mimics (a disjoint variant of) rule (r_1) in the reduction semantics (see Definition 3.2). Constraining the matches to be regular monos ensures that the production is not applied to a graph where nodes y and w_i's are equivalent. This is too restrictive, since a reduction step can be performed if name x coincides with either y or w. Hence, two additional productions are needed: they are variations of p_1^σ, where nodes x_i's are equivalent either to the node y or to the nodes w_i's. We leave these productions unnamed, since they play a minor role in the paper. An analogous production p_2^σ (an the corresponding instances) is needed for rule (r_2).

Correspondingly to rule (r_3) we introduce a production p_3^σ. Note that nodes y and w_i's are already coalesced and the restriction operator is not required, as depicted in Fig. 8. Additionally, an instance where the two names coincide, and the corresponding nodes are thus equivalent, has to be included.

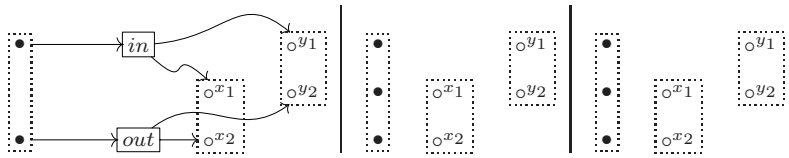

Fig. 8. A production of $\mathcal{G}_\sigma$ encoding reduction (r_3).

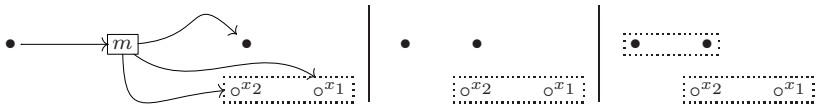

Fig. 9. A production of $\mathcal{G}_\sigma$ encoding reduction (r_4).

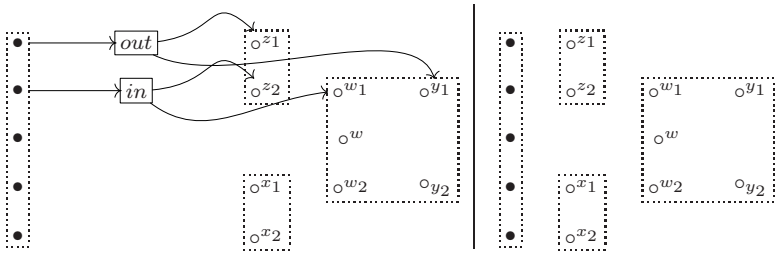

Fig. 10. The e-graphs obtained by applying to $\mathbb{G}_{P_e}$ in Fig. 5(a) first p_1^σ and then p_3^σ.

Finally, production p_4^σ, depicted in Fig. 9, mimics the removal of a match operator as expressed by rule (r_4) in the reduction semantics. The correct application of the rule is ensured by the fact that e-graph morphisms must preserve equivalences.

Observe that, during the reductions, some nodes might be left isolated. These correspond to names that are not referred to anymore by any operator. Hence, a process P is actually related to a class of e-graphs, including $\mathbb{G}_P$ and all the e-graphs which differ from $\mathbb{G}_P$ for the presence of additional isolated nodes.

There is no graphical translation of the context rules since the graph rewriting mechanism allows to apply graph productions in any context (more precisely, a graph production can be applied only if the so-called dangling condition asking that no edge is left dangling is satisfied, but this condition is vacuous here since nodes are never deleted).

3.4 Simulating reductions

Consider again the process $P_e = (\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{y}z)$, and its graphical depiction $\mathbb{G}_{P_e}$ in Fig. 5(a). A possible derivation consists of the two steps below, applying rules r_1 and r_3, respectively:

$$(\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{y}z) \quad \to \quad (w(z) \mid \overline{y}z)\{^y/_w\} = y(z) \mid \overline{y}z \quad \to \quad \mathsf{0}$$

Being the context rules immaterial, we end up by applying to the graph in the left-hand side of Fig. 10 first the rule p_1^σ, and then the rule p_3^σ. The derivation (the derived graphs) is shown in Fig. 10.

Consider now the process $P_m = (\nu w)([y = w].\mathsf{0} \mid x(y) \mid \bar{x}w)$, whose e-graph representation can be found in Fig. 6. Again a derivation consisting of two steps is possible, applying rules (r_1) and (r_4):

$$(\nu w)([y = w].\mathsf{0} \mid x(y) \mid \bar{x}w) \quad \to \quad [w = w].\mathsf{0} \quad \to \quad \mathsf{0}$$

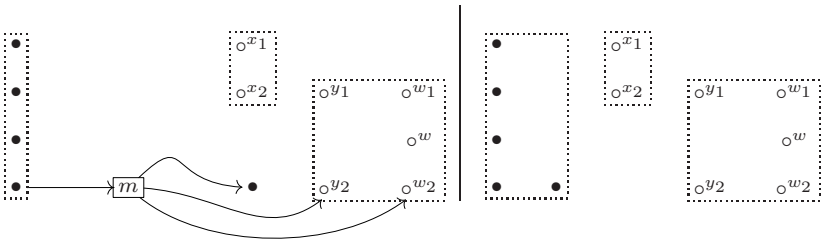

Fig. 11. The e-graphs obtained by applying to $\mathbb{G}_{P_m}$ in Fig. 6 first p_1^σ and then p_4^σ.

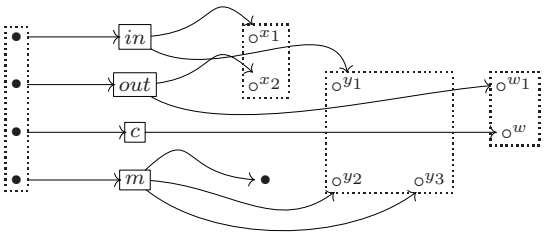

Fig. 12. The e-graph encoding of the process P'_m.

The graphs produced in the corresponding e-graph derivation, using rules p_1^σ and p_4^σ, can be found in Fig. 11.

We next state the sound and complete correspondence between graph derivations and process reduction semantics, after introducing some notation. First, note that for any process P, the graph $\mathbb{G}_P$ has an equivalence class that can be seen as the root of the graph (intuitively, in the quotient, this is the node representing the process P). Then, we say that a rule application to $\mathbb{G}_P$ is a *top* derivation if the image of the match involves a node in the root.

Proposition 3.5 (derivations vs reductions) *Let P be a process. (i) if $P \to Q$, then $\mathcal{G}_\sigma$ entails a top derivation $\mathbb{G}_P \Longrightarrow d$ via a regular mono match, and d is isomorphic to $\mathbb{G}_Q$, up-to the occurrence of isolated nodes. Vice versa, (ii) if $\mathcal{G}_\sigma$ entails a top derivation $\mathbb{G}_P \Longrightarrow d$ via a regular mono match, then there exists a process Q such that $P \to Q$, and d is isomorphic to $\mathbb{G}_Q$, up-to the occurrence of isolated nodes.*

Intuitively, a reduction step is simulated by applying an enabled production, i.e., by finding a match covering a sub-graph containing a root node. The "garbage collection" phase is needed in order to remove all those name nodes that are not referred to anymore by an operator. Similarly, the restriction to top derivations forces the match to be applied to top operators, forbidding the occurrence of a reduction inside the outermost match operators. This constraint could be "embedded" in the graphical representation by using a special mark (e.g., a "go" edge) which is required and then propagated by the graph rewriting rules (see, e.g., [8,9]).

In the paper [1] a partial order concurrent semantics on rewrites is developed for graphs with equivalences, which is thus inherited by process calculi encoded in this formalism. A detailed description of these aspects is outside the scope of this paper. Let us only notice that in the derivation considered for process P_m (see Fig. 11) the two steps are causally related. In fact, the production p_4^σ requires the nodes y_2 and w_2 to be equivalent, and such equivalence is generated by the application of p_1^σ.

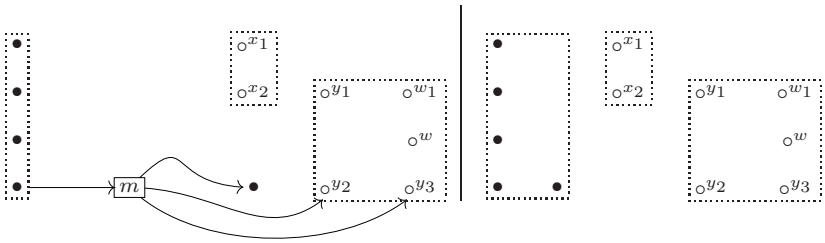

Fig. 13. The e-graphs obtained by applying to $\mathbb{G}_{P'_m}$ in Fig. 12 first p_1^σ and then p_4^σ.

Consider now the process $P'_m = (\nu w)([y = y].\mathtt{0} \mid x(y) \mid \bar{x}w)$, which differs from the process P_m because the match is already satisfied, and its graphical representation in Fig. 12. The same sequence of rule applications as for the derivation depicted in Fig. 11 can now be replicated, and the result (the derived graphs) is presented in Fig. 13. With respect to the derivation in Fig. 11, production p_4^σ now reads the equivalence class containing $\{y_2, y_3\}$, which is in the initial graph, and thus no casual dependency holds between the production occurrences. Formally, the components of the derivation are sequentially independent, since the coalescing of nodes corresponding to the names y and w is not needed for the second direct derivation.

4 Conclusions and further works

This paper discusses the expressiveness of a novel formalism for the analysis of distributed systems, *graphs with equivalences*: (hyper-)graphs equipped with an equivalence relation over their nodes. The formalism has been introduced in [1], and it has been shown there to be amenable to the usual tools of the DPO approach to graph transformation (in particular, the theoretical results associated to the concurrent features of the approach). In this paper we test the formalism against the encoding of a simple process calculus with name mobility, the solo calculus, proving it suitable for the purpose of simulating the reduction semantics of the calculus.

An obvious strand of research we are currently pursuing it to properly establish the connection between the category of graphs and of graphs with equivalences, making precise the correspondence briefly hinted at in Section 2. On a related note, observe that e-graphs resemble the so-called *structures*, as defined in [6]. Indeed, along the same lines of [12, Section 6], the category **EGraph** should be proved *quasi-adhesive*, thus inheriting part of the rich theory developed for such formalism.

References

[1] P. Baldan, F. Gadducci, and U. Montanari. Concurrent rewriting for graphs with equivalences. In C. Baier and H. Hermanns, editors, *Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2006.

[2] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.

[3] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[4] 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 Rozenberg [16], pages 163–245.

[5] F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In Rozenberg [16], pages 95–162.

[6] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1:361–404, 1991.

[7] H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. III: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[8] F. Gadducci. Term graph rewriting and the π-calculus. In A. Ohori, editor, *Programming Languages and Semantics*, volume 2895 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2003.

[9] F. Gadducci and U. Montanari. A concurrent graph semantics for mobile ambients. In S. Brookes and M. Mislove, editors, *Mathematical Foundations of Programming Semantics*, volume 45 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2001.

[10] F. Gadducci and U. Montanari. Graph processes with fusions: concurrency by colimits, again. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods (Ehrig Festschrift)*, volume 3393 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2005.

[11] P. Gardner and L. Wischik. Explicit fusion. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2000.

[12] S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Informatique Théorique et Applications/Theor. Informatics and Applications*, 39:511–545, 2005.

[13] C. Laneve and B. Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13:675–683, 2002.

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

[15] J. Parrow and B. Victor. The fusion calculus: Expressiveness and simmetry in mobile processes. In V. Pratt, editor, *Logic in Computer Science*, pages 176–185. IEEE Computer Society Press, 1998.

[16] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. I: Foundations*. World Scientific, 1997.