Concurrent Rewriting for Graphs with Equivalences*

Paolo Baldan[1], Fabio Gadducci[2], and Ugo Montanari[2]

[1] Dipartimento di Scienze dell'Informazione, Università Ca' Foscari di Venezia
[2] Dipartimento di Informatica, Università of Pisa

Abstract. Several applications of graph rewriting systems (notably, some encodings of calculi with name passing) require rules which, besides deleting and generating graph items, are able to coalesce some parts of the graph. This latter feature forbids the development of a satisfactory concurrent semantics for rewrites (intended as a partial order description of the steps in a computation). This paper proposes the use of *graphs with equivalences*, i.e., (typed hyper-) graphs equipped with an equivalence over nodes, for the analysis of distributed systems. The formalism is amenable to the tools of the double-pushout approach to rewriting, including the theoretical results associated to its concurrent features. The formalism is tested against the encoding of a simple calculus with name mobility, namely the *solo calculus*.

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 modeling 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 (as witnessed by [6] and the many areas where it found applications), abstracting away from the often unnecessary details of the state representation. As an example, DPO rewriting techniques for simulating reductions in nominal calculi [17, 4], as presented in [9, 10], views a (possibly recursive) process as a graph, thus modeling reductions by rewrites. The use of graphs allows for getting rid of the problems concerning the implementation of reduction over the structural congruence, such as e.g. α-conversion of (bound) names, since equivalent processes turn out to be mapped into isomorphic graphs.

* Research partially supported the EC RTN 2-2001-00346 SegraVis, the EU IST-2004-16004 SEnSOria and the MIUR Project ART.

However, the widespread diffusion of the formalism raises unresolved issues concerning the analysis of its concurrency aspects. Consider again the graphical encodings for nominal calculi we 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).

Such features are general enough to deserve to be properly addressed. Recall that concurrency in the DPO approach was originally defined by using the *shift equivalence* [6], which equates those derivations that could be related via the repeated application of an interchange operator swapping consecutive rewriting steps that are *sequentially independent* (roughly, such that they act on disjoint parts of the graph). *Graph processes*, as proposed in [5], generalise the notion of non-sequential process from the Petri net mold, representing concurrency and causal dependency in a synthetic manner as a partial ordering on the rewrites occurring in a derivation. Shift equivalent derivations correspond to isomorphic processes. Additionally, each total order on rule instances, compatible with the partial order of the graph process, uniquely characterises a derivation which is shift equivalent to the original one [1] (*complete concurrency* property). The above theory has been developed for rules with injective right-hand morphisms. When considering coalescing rules, as argued in [14, 11], a connection between graph processes and shift-equivalent derivations may still be drawn, but no partial order can be distilled anymore from a graph process.

In order to allow the use of coalescing rules, while retaining a satisfactory theory of concurrency, we advocate the use of rewriting over a novel family of structures, *graphs with equivalences*, which are 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 for recovering the theoretical results associated to the concurrent features of the DPO approach: the paradigm of graph processes for representing shift-equivalent derivations can be lifted to the new formalism, and the complete concurrency property once more holds.

For the sake of presentation, the formalism is tested against the encoding of (a fragment of the) *solo calculus* [16], one of the dialects of those nominal calculi whose distinctive feature is name fusion [12, 18]. The choice of such a simple calculus is functional to the main focus of the paper, but it is noteworthy that the formalism is expressive enough to properly recast the graphical encodings of nominal calculi proposed in e.g. [9, 10]. With respect to those encodings, where the presence of node coalescing rules forbade the development of a suitable concurrent presentation of reductions, the use of equivalences on nodes allows the extraction of a meaningful notion of causal order from a process.

The paper has the following structure. In Section 2 we introduce the formalism of graphs with equivalences, which is proved to be amenable to the DPO approach to rewriting. In Section 3 we develop a concurrency theory for rewriting of graphs with equivalences. Section 4 presents an encoding of the solo calculus into graphs with equivalences, showing how it allows for an analysis of its concurrency properties. Finally, Section 5 concludes the paper, discussing open issues and directions of future research.

2 Rewriting Graphs with Equivalences

In this section we introduce the category of graphs with equivalences, which are graphs endowed with an equivalence over the set of nodes. Rewriting systems over such structures are proposed as a technically convenient replacement of rewriting over ordinary graphs where rules may coalesce nodes.

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. An (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 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 $\mathbb{G}$ is intended to provide 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 $\mathcal{Q}(f)([v]_{\sim_G}) = [f(v)]_{\sim_G}$.

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**. This is easily proved observing that **Graph** is equivalent to the full subcategory of **EGraph** where objects are e-graphs with all non-equivalent nodes (i.e., e-graphs $\mathbb{G}$ where $\sim_G$ is the identity). *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.

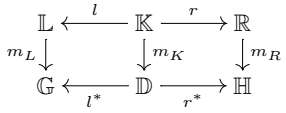

Fig. 1. A direct derivation.

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 [6, 7]. 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. [5] 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.

Definition 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.*

Observe that, given a left-linear production p, in the graph 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 some nodes.

Definition 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, the fact that l is 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 a 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, e.g., in the encoding of nominal calculi, it is necessary to consider injective matches only. When dealing with e-graphs, this property corresponds to the requirement of having regular mono matches. 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 in the standard approach can be represented by an equivalence class of possibly unbounded size. Therefore, in order to model node deletion, also an unbounded number of rules deleting equivalence classes of arbitrary size must be inserted into a transformation system. However, notice that for modelling purposes, it is often not restrictive to consider only rules which never delete nodes: indeed, this is what happens on most graphical encodings of process calculi. Node deletion is then simulated by leaving a node isolated, thus assuming an implicit mechanism for performing garbage collection.

3 Concurrency in E-Graph Rewriting

In this section we show that the notion of sequential independence, characterising independent steps in a computation, may be extended to the setting of e-graphs. More importantly, also the notion of process may be generalised, thus providing a partial order description of concurrency in computations: a generalization that fails when considering standard graphs with coalescing rules.

3.1 Sequential independence and Shift-equivalence

The notion of sequential independence is aimed at characterising direct derivations which do not interfere with each other and thus which could be potentially applied in any order (and concurrently). The definition below, a stronger version of the standard one, is inspired to the notion proposed in [14] for DPO rewriting with injective matches.

Definition 4 (sequential independence). *Let* $\mathbb{G} \overset{p_1/m_1}{\Longrightarrow} \mathbb{H} \overset{p_2/m_2}{\Longrightarrow} \mathbb{M}$ *be a derivation as in Fig. 2. Then, its components are* sequentially independent *if there exists an* independence pair *among them, i.e., two e-graph morphisms* $i_1 : \mathbb{R}_1 \to \mathbb{D}_2$ *and* $i_2 : \mathbb{L}_2 \to \mathbb{D}_1$ *such that* $l_2^* \circ i_1 = m_{L_2}$, $r_1^* \circ i_2 = m_{R_1}$ *and* $r_2^* \circ i_1$ *is regular mono.*

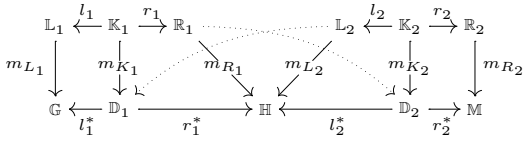

Fig. 2. (Strong) sequential independence for derivation $\rho = \mathbb{G} \overset{p_1/m_1}{\Longrightarrow} \mathbb{H} \overset{p_2/m_2}{\Longrightarrow} \mathbb{M}$.

Requiring $r_2^* \circ i_1$ to be regular mono is motivated by the interplay between the equivalences the application of a rule may produce and the request for the matches to be regular mono. Roughly, the second direct derivation must not equate items which are read by the first one: otherwise, the application of the two productions could not be swapped, keeping the matches regular mono.

Proposition 5 (interchange operator). *Let $\rho = \mathbb{G} \overset{p_1/m_1}{\Longrightarrow} \mathbb{H} \overset{p_2/m_2}{\Longrightarrow} \mathbb{M}$ be a derivation, and let its components be sequentially independent via an independence pair ξ. Then, a derivation $IC_\pi(\rho) = \mathbb{G} \overset{p_2/m_2^*}{\Longrightarrow} \mathbb{H}^* \overset{p_1/m_1^*}{\Longrightarrow} \mathbb{M}$ can be uniquely chosen, such that its components are sequentially independent via a canonical independence pair ξ^*.*

The interchange operator can be used to formalise a notion of shift-equivalence [6], identifying (as for the analogous, better-known *permutation equivalence* of λ-calculus) those derivations which differ only for the scheduling of independent steps. This equivalence abstracts also from the concrete identity of items involved in a derivation, i.e., it considers derivations up-to isomorphism (defined component-wise, in the obvious way).

Definition 6 (shift-equivalence). *Two derivations ρ and ρ' are* shift-equivalent*, written $\rho \equiv_s \rho'$, if repeatedly applying the interchange operator to ρ we can obtain a derivation isomorphic to ρ'.*

The shift-equivalence class $[\rho]_s$ of a derivation ρ can be considered as a representation of a concurrent derivation which abstracts from the order of non-interfering rewriting steps.

3.2 Processes for e-graphs

A more concrete, yet equivalent notion of abstract derivation for an e-GTS is obtained by generalising the so-called *graph process semantics* [1]. As for the similar notion on Petri nets [13], a *graph process* is aimed at describing a derivation abstracting away from the ordering of causally unrelated steps, and thus it offers at the same time a concrete representative for a class of shift-equivalent derivations. We will see that, differently from what happens in the case of graph transformation systems with coalescing rules, the notion of process for e-graphs provides a faithful partial order representation of concurrency in a derivation.

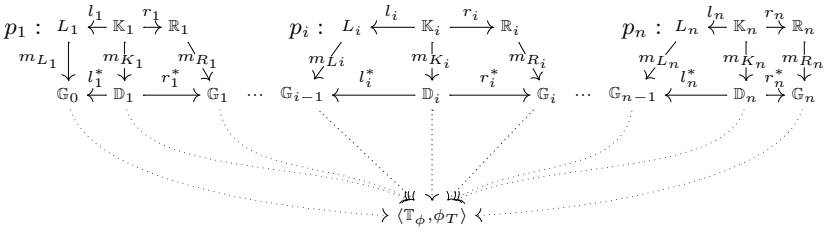

Fig. 3. Colimit construction for derivation $\rho = \mathbb{G}_0 \overset{p_1/m_1}{\Longrightarrow} \ldots \overset{p_n/m_n}{\Longrightarrow} \mathbb{G}_n$.

Definition 7 (e-graph process). *Let $\mathcal{G}$ be an e-*GTS *and $\rho = \mathbb{G}_0 \overset{p_1/m_1}{\Longrightarrow} \ldots \overset{p_n/m_n}{\Longrightarrow} \mathbb{G}_n$ a derivation (upper part of Fig. 3). The* e-graph process *associated to ρ is a tuple $\phi = \langle \mathcal{O}_\phi, \phi_T, \phi_P, \mathbb{I}, \mathbb{F} \rangle$, where $\mathcal{O}_\phi = \langle \mathbb{T}_\phi, P_\phi, \pi_\phi \rangle$ is an e-*GTS *and $\phi_T : \mathbb{T}_\phi \to \mathbb{T}$ is an e-graph morphism and $\phi_P : P_\phi \to P$ is a function, defined as*

- *$\langle \mathbb{T}_\phi, \phi_T \rangle$ is a colimit object (in $\mathbb{T}$-***Graph***) of the diagram representing derivation ρ, as depicted in Fig. 3;*
- *$P_\phi = \{\langle p_j, j \rangle \mid j \in \{1, \ldots, n\}\}$. For all j, $\pi_\phi(\langle p_j, j \rangle)$ is essentially the production p_j, but retyped over $\mathbb{T}_\phi$ by the morphisms uniquely induced by the colimit (see Fig. 3). Moreover, $\phi_P(\langle p_j, j \rangle) = p_j$;*
- *$\mathbb{I}$ and $\mathbb{F}$ are the graphs $\mathbb{G}_0$ and $\mathbb{G}_n$, typed over $\mathbb{T}_\phi$ by the morphisms induced by the colimit. They are called* source *and target of the process and denoted $src(\Pi(\rho))$ and $trg(\Pi(\rho))$.*

The process associated to a derivation ρ, as defined above, is denoted by $\Pi(\rho)$.

The colimit construction applied to a derivation ρ essentially constructs the type graph as a copy of the source graph plus the items created during the derivation. Productions are instances of production applications. Additionally, the colimit operation "collects" the generated equivalences: the equivalence on the e-graph arising as type graph of $\Pi(\rho)$ is the "union" of the equivalences of the graphs occurring in ρ.

It can now be shown that two derivations are shift equivalent iff the corresponding processes are isomorphic, and thus processes properly capture the notion of concurrency as expressed by shift-equivalence.

Proposition 8 (Shift equivalence vs processes). *Let ρ and ρ' be derivations. Then $\rho \equiv_s \rho'$ if and only if the processes $\Pi(\rho)$ and $\Pi(\rho')$ are isomorphic.*

The result above is standard in graph rewriting theory for rules where both morphisms are monos. It was generalized to strict derivations and rules coalescing nodes in [11, Thms 1–2]. However, in that setting it was impossible to provide a technique for extracting from a process any information about the dependencies between the single direct derivations occurring in it (see [11, Section 4.2]).

3.3 Full concurrency for e-graph processes

In order to extract from a process ϕ sound information about the dependencies between events, as for rewriting over ordinary graphs, we define the pre-set, post-set and context of a production, which roughly identify the items which are deleted, produced and preserved by a production.

Definition 9 (pre-set, post-set, context). *Let $\phi = \langle \mathcal{O}_\phi, \phi_T, \phi_P, \mathbb{I}, \mathbb{F} \rangle$ be a process, where $\mathcal{O}_\phi = \langle \mathbb{T}_\phi, P_\phi, \pi_\phi \rangle$. For any $p \in P_\phi$ we define*

$$^\bullet p = t_{\mathbb{L}_p}(|\mathbb{L}_p| - l_p(|\mathbb{K}_p|)) \qquad p^\bullet = t_{\mathbb{R}_p}(|\mathbb{R}_p| - r_p(|\mathbb{K}_p|)) \qquad \underline{p} = t_{\mathbb{K}_p}(|\mathbb{K}_p|)$$

considered as sets of nodes and edges, and we say that p consumes*,* produces *and* preserves *items in $^\bullet p$, $p^\bullet$ and $\underline{p}$, respectively.*

The mutual relationships between the pre-sets, post-sets and contexts of productions naturally lead to a precedence relation between the productions in a process (generalising to e-graphs the asymmetric conflict relation [3]).

Definition 10 (precedence relation). *Let ϕ be a process as in Def. 9. The* precedence relation *is the binary relation $\nearrow_\phi$ over the set P_ϕ of productions, defined by $p \nearrow_\phi p'$ if (1) $p^\bullet \cap (^\bullet p' \cup \underline{p'}) \neq \emptyset$ and $p \neq p'$ or (2) $\underline{p} \cup \cap ^\bullet p' \neq \emptyset$.*

Observe that when p' uses something produced by p necessarily p' follows p (point 1). Similarly, when p' consumes an item read by p, the only possible order of execution is p followed by p' (point 2).

However, $\nearrow_\phi$ alone does not suffice to faithfully mirror the relationship between productions since additional dependencies arise whenever productions "read" equivalences among nodes and generate new ones. Hence, we now characterise the equalities between nodes needed and generated by any production.

Definition 11 (read and produced equivalences). *Let ϕ be a process as in Def. 9. For any $p \in P_\phi$ we define*

$$req(p) = t_{\mathbb{L}_p}(\sim_{L_p}) \qquad \text{and} \qquad grel(p) = t_{\mathbb{R}_p}(\sim_{R_p} - t_{\mathbb{K}_p}(\sim_{K_p}))$$

and call them the (symmetric) relations read and produced by p. Given a set of productions $X \subseteq P_\phi$ we write $geq(X)$ for the set $(\bigcup_{p \in X} grel(p) \cup \sim_{src(\phi)})^*$.

Note that since all matches are regular monos, the application of a production never generates an already existing equivalence. This implies that any equivalence between nodes has a uniquely determined history, whose events are thus causes for productions which read that equivalence.

Proposition 12 (generating relation). *Let ϕ be a process as in Def. 9. Then for any production $p \in P_\phi$ there exists a least subset of productions $eq(p) \subseteq P_\phi$ such that $req(p) \subseteq geq(eq(p))$.*

Now, all events in $eq(p)$ must precede p in the computation, as expressed by the relation defined below.

Definition 13 (e-precedence relation). *Let ϕ be a process as in Def. 9. The* e-precedence relation *is the binary relation $\nearrow^e_\phi$ over the set P_ϕ of productions, defined by*

$$\nearrow_\phi \cup (\bigcup_{p \in P_\phi} eq(p) \times \{p\})$$

Then it can be shown that relation $\nearrow^e_\phi$ faithfully captures the dependencies between events in a process, i.e., we can prove the following result.

Proposition 14 (full concurrency). *Let ϕ be a process. Then the productions of ϕ, applied to in any order compatible with $\nearrow^e_\phi$, rewrite $src(\phi)$ into $trg(\phi)$ and all such derivations are shift equivalent.*

This "permutation" result does not hold for graph rewriting rules that may coalesce nodes, hence the notion of process fails to work when dealing with standard graphs. The reason for this failure is due to the fact that node identity is lost after a fusion, while node equivalences allow for a natural way of taking into account these additional dependencies.

4 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* [16], one of the dialects of those nominal calculi whose distinctive feature is name fusion [12, 18]. We will see that the tools introduced in the previous section, like shift-equivalence and process semantics, allow for providing a characterisation of concurrent reductions in the solo calculus.

4.1 The monadic fragment of the solo calculus

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

Definition 15 (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*

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

The operators $x(y)$ and $\overline{x}y$ are denoted as *input* and *output*, respectively, even if their symmetric behaviour makes the distinction (typical instead of other calculi) immaterial; collectively, each instance of them is called a *solo*, to emphasise its lack of connections, except for some possible name sharing, with the other operators. 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.

$$P \mid Q = Q \mid P \qquad P \mid \mathtt{0} = P \qquad P \mid (Q \mid R) = (P \mid Q) \mid R$$

$$(\nu x)(\nu y)P = (\nu y)(\nu x)P \qquad (\nu x)\mathtt{0} = \mathtt{0} \qquad (\nu x)(P \mid Q) = P \mid (\nu x)Q \quad \text{for } x \notin \mathit{fn}(P)$$

Fig. 4. The set of structural axioms.

Definition 16 (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 equations in Fig. 4, generated by the following inference rules*

$$(r_1)\ \frac{y \neq w}{(\nu w)(x(y) \mid \overline{x}w \mid P) \to P\{^y/_w\}} \qquad (r_2)\ \frac{y \neq w}{(\nu y)(x(y) \mid \overline{x}w \mid P) \to P\{^w/_y\}}$$

$$(r_3)\ \frac{}{x(y) \mid \overline{x}y \to \mathtt{0}} \qquad (r_4)\ \frac{P \to Q}{(\nu x)P \to (\nu x)Q} \qquad (r_5)\ \frac{P \to Q}{P \mid R \to Q \mid R}$$

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

The top rules characterise the communication between restricted processes. Consider the second: the process $\overline{x}w$ is ready to communicate the 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, the communication has a global effect on the process as a whole. Note that one of the names among $\{y, w\}$ *has to be bound*, so that, in principle, the rule does not to alter the number of free names floating around and the possible choice requires the presence of two different rules. The third rule expresses the fact that there is no reason to bind a name during a reduction, if no substitution has actually to occur. The latter two rules simply state the closure of the reduction relation with respect to the operators of restriction and parallel composition.

The axioms for structural congruence in Fig. 4 state that a process is a collection of solos floating around, and interacting by forcing some name fusion. The only difference with respect to the monadic fragment of the calculus proposed in [16] is the lack of a *match* operator $[x = y]$, avoided to simplify the presentation, and the explicit presentation of the three reduction rules, which in [16] are summarised as a unique rule equipped with constraints on the substitution induced by the name fusion.

4.2 The graphical encoding of solos

This section informally presents an encoding of solos based on e-graphs. It resembles the encoding using standard graphs presented in [11, Section 5], basically replacing node coalescing rules with rules generating node equivalences. Its formal definition is not presented for space limitation: it is easily obtained by adapting the proposals for mobile ambients and π-calculus in [9, 10].

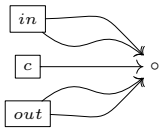

Fig. 5. The type graph T_σ.

In order to help intuition, we begin with a description of a suitable normal form for structurally congruent processes. First notice that any process P is equivalent to a process of the shape $(\nu x_1)\dots(\nu x_n)(\sigma_1 \mid \dots \mid \sigma_m)$ where all x_i's are different, all σ_j's are solos, and the set $X = \{x_1 \dots x_n\}$ contains only names occurring in $S = \sigma_1 \mid \dots \mid \sigma_m$, that is, $X \subseteq \mathtt{fn}(S)$. Thus we can denote a process in normal form as $(\nu X)\mathcal{P}$, for $\mathcal{P}$ a set of solos, since the order of the restriction operators and of the solos is immaterial.

Definition 17 (disjoint normal form). *Let P be a solo process and let $(\nu X)\mathcal{P}$ be its normal form. We call* disjoint normal form *of P an expression of the kind $(\nu X)\mathcal{D}\xi$, where $\mathcal{D}$ is a set of solos with disjoint names such that $X \cap \mathit{fn}(\mathcal{D}) = \emptyset$ and $\xi : \mathit{fn}(\mathcal{D}) \to \mathit{fn}(\mathcal{P})$ is a surjective name substitution satisfying $\mathcal{D}\xi = \mathcal{P}$.*

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

The above characterisation naturally suggests a representation using typed e-graphs. The type e-graph $\mathbb{T}_\sigma$, represented in Fig. 5, has one node and three 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 will be represented by labelling graph edges with *in*, *out* and *c*.

Let P be a process, and $(\nu X)\mathcal{D}\xi$ its disjoint normal form. Then the typed e-graph $\mathbb{G}_P$ associated to P has as many edges and nodes as operators and names, respectively, occurring in $\mathcal{D}$. The effect of the substitution ξ is represented by using the equivalence $\sim_{G_P}$ between nodes: given two nodes x and y we have $x \sim_{G_P} y$ iff $\xi(x) = \xi(y)$ or $\xi(x) = y$. So, 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. 6(a), where nodes, for the sake of clarity, are equipped with the name they represent. 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 $\{w, w_1, w_2\}$. Some intuition may be gained by looking at the graph $\mathcal{Q}(\mathbb{G}_{P_e})$ depicted in Fig. 6(b), obtained by collapsing equivalent nodes (this was indeed the encoding proposed for process P_e in [11, Fig. 11]).

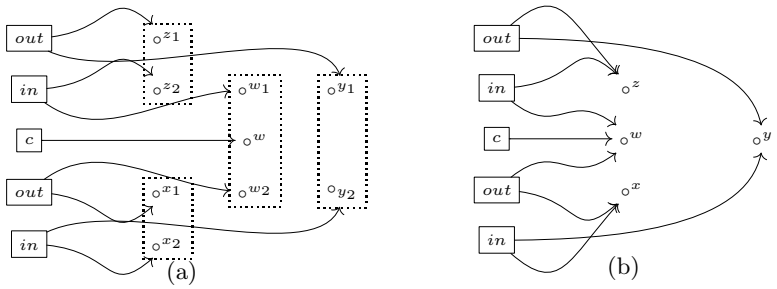

Fig. 6. (a) The e-graph $\mathbb{G}_{P_e}$ encoding a process P_e and (b) the quotient graph $\mathcal{Q}(\mathbb{G}_{P_e})$.

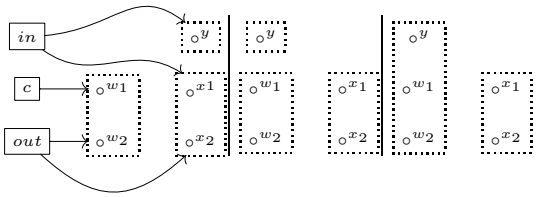

Fig. 7. The first production of $\mathcal{G}_\sigma$.

4.3 Encoding reductions

We now introduce the e-GTS $\mathcal{G}_\sigma$ in $\mathbb{T}_\sigma$-**EGraph**, showing how it simulates the reduction semantics for solo processes. It it basically contains just three productions (i.e., one for each axiom 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 action of the rule is described by the names of the nodes: as an example, the nodes identified by y and w_i's, distinct in $\mathbb{L}_1^\sigma$, are made equivalent in $\mathbb{R}_1^\sigma$. The node identifiers are of course arbitrary: they are used just to characterise the span of morphisms.

The rule mimics (a disjoint variant of) the first axiom of the reduction semantics, as given in Def. 16. 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. Nevertheless, this turns out to be 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.

A similar situation occurs when the name y on the input operator, instead of the name w on the output operator, is bound: it suffices a production p_2^σ (together with two instances) mirroring p_1^σ. Most important, a production p_3^σ is needed, where 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.

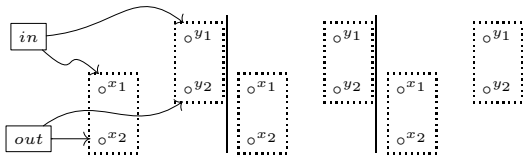

Fig. 8. The third production of $\mathcal{G}_\sigma$.

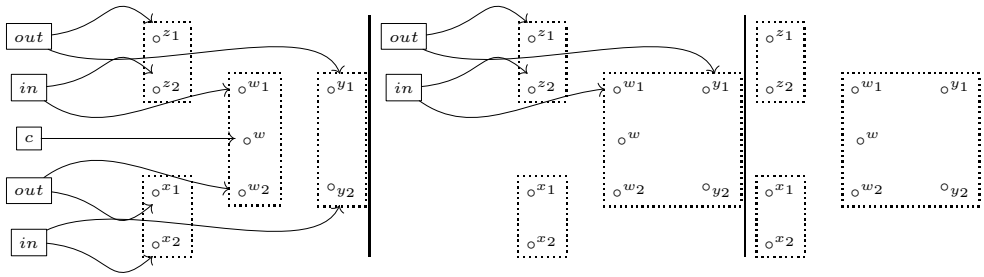

Fig. 9. The derived graphs of a derivation first applying p_1^σ and then p_3^σ.

Observe that, during the reductions, isolated nodes may arise in correspondence of unused names. Hence, in the encoding a process P actually corresponds to a class of e-graphs, including $\mathbb{G}_P$, as defined in the previous section, and all the e-graphs which differ from $\mathbb{G}_P$ for the presence of additional isolated nodes.

4.4 Concurrency via fusion

Consider the process $(\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{y}z)$, and its graphical depiction $\mathbb{G}_{P_e}$ in Fig. 6(a). A possible derivation consists of the two steps below, applying rules r_1 and r_3, respectively.

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

Being the context rules immaterial, we end up by applying to the graph in the left-hand side of Fig. 9 first the rule p_1^σ, and then the rule p_3^σ. The derivation (the derived graphs) is shown in Fig. 9, and the associated process is in Fig. 10.

It can be easily seen that the two steps are *not* sequentially independent. This is indeed recorded in the process $\Pi(\rho)$, as depicted in Fig. 10. The production p_1^σ consumes three edges, reads three equivalence classes (namely, those for nodes $\{w, w_2\}$, x_i's and y_i's), and generates the symmetric relation containing $\{\langle w, y_j\rangle, \langle w_i, y_j\rangle \mid i, j = 1, 2\}$. For the sake of readability, productions have dotted arrows only to (the smallest) equivalence relations including $\sim_{src(\rho)}$ that they read or generate. Now, p_3^σ reads the class $\{w_1, y_1\}$, so that $req(p_3^\sigma)$ is contained in $geq(\{p_1^\sigma\})$: thus, differently from what happens considering just the relation $\nearrow$, here $p_1^\sigma \nearrow^e p_3^\sigma$, i.e., the dependency between (the applications of) the production p_1^σ and the production p_3^σ is properly recorded.

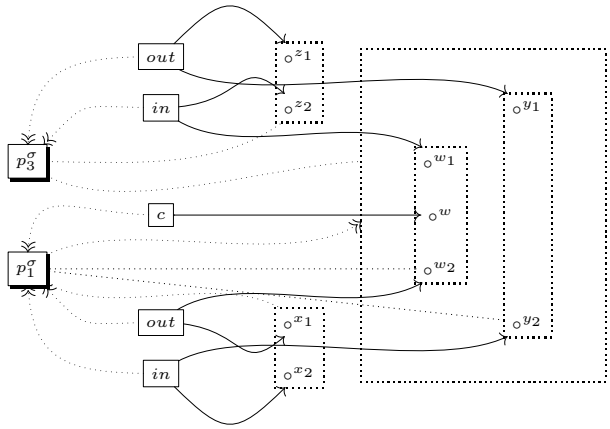

Fig. 10. The process of the derivation in Fig. 9.

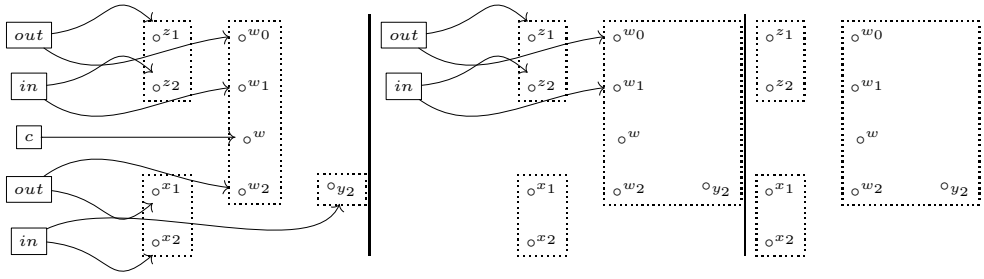

Fig. 11. The derived graphs of another derivation first applying p_1^σ and then p_3^σ.

Let us now consider the process $(\nu w)(x(y) \mid \overline{x}w \mid w(z) \mid \overline{w}z)$, which differs from the process above just for the name occurring in the right-most solo (namely, $\overline{w}z$ instead of $\overline{y}z$). The same sequence of rule applications as for the derivation depicted in Fig. 9 can now be replicated, and the result (the derived graphs) is presented in Fig. 11. The process $\Pi(\rho')$ is depicted in Fig. 12: with respect to the process in Fig. 10, production p_3^σ now reads the equivalence class containing $\{w_0, w_1\}$, instead of the class containing the w, w_i's and y generated by p_1^σ: thus, $req(p_3^\sigma)$ is contained in $\sim_{src(\rho')}$, and no casual dependency holds between the production occurrences. Hence the components of the derivation are sequentially independent, since the coalescing of nodes w, w_i's and y is not needed for the second direct derivation.

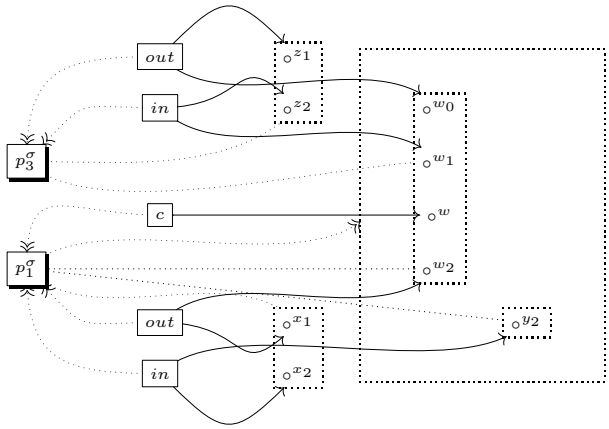

Fig. 12. The process of the derivation in Fig. 11.

5 Conclusions and further works

The paper introduces a novel formalism for the analysis of distributed systems, *graphs with equivalences*: typed (hyper-)graphs equipped with an equivalence relation over their nodes. The formalism is 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 (the paradigm of graph processes for representing shift-equivalent derivations) can be lifted to the new formalism.

We are planning two related strands of research. On the one side, we would like to properly establish the connection between the category of graphs and of graph with equivalences, making precise the correspondence briefly hinted at in Section 2. On the other side, we need to further develop the theory surrounding the graph process construction, drawing a link with respect to a suitable notion of event structure, amenable to model non-determinism in derivations. The latter characterisation would provide a further sanity check, providing a concurrent semantics for nominal calculi, to be compared with already existing proposals.

As a final remark, observe that e-graphs resemble the so-called *structures*, as defined in [8]. Indeed, along the same lines of [15, Section 6] that the category **EGraph** can be proved *quasi-adhesive*, thus inheriting part of the rich theory developed for such formalism. That very same paper develops a general theory of DPO rewriting for (quasi-)adhesive categories and a theory of processes is proposed in [2]. Unfortunately, this could not be helpful for our purposes, since the use of rules where right-hand side morphisms are not regular monos makes a relevant part of such theory not applicable.

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, T. Heindel, B. König, and P. Sobociński. Processes for adhesive rewriting systems. In L. Aceto and A. Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures*, volume 3921 of *Lect. Notes Comp. Sc.*, pages 202–216. Springer, 2006.
3. P. Baldan, A. Corradini, and U. Montanari. Unfolding and event structure semantics for graph grammars. In W. Thomas, editor, *Foundations of Software Science and Computation Structures*, volume 1578 of *Lect. Notes Comp. Sc.*, pages 73–89. Springer, 1999.
4. L. Cardelli and A. Gordon. Mobile ambients. *Th. Comp. Sc.*, 240:177–213, 2000.
5. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
6. 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, pages 163–245. World Scientific, 1997.
7. 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, pages 95–162. World Scientific, 1997.
8. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Str. Comp. Sc.*, 1:361–404, 1991.
9. F. Gadducci. Term graph rewriting and the π-calculus. In A. Ohori, editor, *Programming Languages and Semantics*, volume 2895 of *Lect. Notes Comp. Sc.*, pages 37–54. Springer, 2003.
10. 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 *El. Notes Th. Comp. Sc.* Elsevier Science, 2001.
11. 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 *Lect. Notes Comp. Sc.*, pages 84–100. Springer, 2005.
12. P. Gardner and L. Wischik. Explicit fusion. In M. Nielsen and B. Rovan, editors, *Mathematical Foundations of Computer Science*, volume 1893 of *Lect. Notes Comp. Sc.*, pages 373–382. Springer, 2000.
13. U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
14. A. Habel, J. Müller, and D. Plump. Double-pushout graph transformation revisited. *Math. Str. Comp. Sc.*, 11:637–688, 2001.
15. S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Informatique Théorique et Applications/Theor. Informatics and Applications*, 39:511–545, 2005.
16. C. Laneve and B. Victor. Solos in concert. *Math. Str. Comp. Sc.*, 13:675–683.
17. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. Part I and II. *Information and Computation*, 100:1–77, 1992.
18. 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.