An Inductive View of Graph Transformation*

F. Gadducci[1] and R. Heckel[2]

[1] TUB, Fachbereich 13 Informatik, Franklinstraße 28/29, 10587 Berlin, Germany, (gfabio@cs.tu-berlin.de).
[2] Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56124 Pisa, Italy, (reiko@di.unipi.it).

Abstract. The dynamic behavior of rule-based systems (like *term rewriting systems* [24], *process algebras* [27], and so on) can be traditionally determined in two orthogonal ways. Either *operationally*, in the sense that a way of embedding a rule into a state is devised, stating explicitly how the result is built: This is the role played by (the application of) a substitution in term rewriting. Or *inductively*, showing how to build the class of all possible reductions from a set of basic ones: For term rewriting, this is the usual definition of the rewrite relation as the minimal closure of the rewrite rules. As far as *graph transformation* is concerned, the operational view is by far more popular: In this paper we lay the basis for the orthogonal view. We first provide an inductive description for graphs as arrows of a freely generated *dgs-monoidal* category. We then apply 2-categorical techniques, already known for term and term graph rewriting [29, 7], recasting in this framework the usual description of graph transformation via double-pushout [13].

1 Introduction

The *theory of graph transformation* [30] basically studies a variety of formalisms which extend the theories of formal languages and term rewriting, respectively, in order to deal with structures more general than strings and terms. In both of these "classical" rewrite formalisms, there are two different ways of defining the rewrite relation $\rightarrow_{\mathcal{R}}$ for a given rewrite system $\mathcal{R}$: The *operational definition* states e.g. that a term rewrite rule $l \rightarrow r \in \mathcal{R}$ is applicable to a term t if an instance of l occurs as a subterm in t. Then, this subterm may be removed and replaced by a corresponding instance of r, leading to a derived term s. Equivalently, we may give an *inductive definition* where the rewrite relation is the smallest relation which contains $\mathcal{R}$ and is closed under substitution and context. While the operational definition is clearly more intuitive, and well-suited for implementation purposes, the inductive one plays an important role in the theory of term rewriting, since it allows for the development of analysis techniques based on structural induction.

* Research partly supported by the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems) through the Technical University of Berlin and the University of Pisa.

In the *double-pushout* (DPO) approach to graph transformation [15, 13] (and in most of the other approaches) the operational definition is by far more popular. Inductive definitions of DPO graph transformation have been given in [1, 12], but they do not have the same role as in the theory of term rewriting. One reason may be that, unlike for strings and terms, there is no straightforward inductive definition of graphs. Rather, each possible interpretation suggests a different choice of the basic operations. In [1], for example, a hyper-graph is considered as *a set of edges glued by means of vertices*. The operations for building graphs are disjoint union, and renaming and fusion of nodes. In [12], a graph is described in a logical style by *a set of edge predicates over node variables*, using (partly non-disjoint) union, and renaming and binding of variables.

In this paper, graphs are seen as *distributed states* consisting of several local components connected through interfaces. The distributed structure is made explicit by regarding a graph as an *arrow* of a category whose objects are sets of interface vertices: The source and target object of an arrow representing a local state are the interfaces through which it is connected to other local states. Then, arrow composition represents the composition of two local states over a common interface. Our inductive definition of graphs is thus based on the most basic operation in the DPO approach: The *gluing of graphs*, categorically described as a pushout in the category of graphs. This category of *ranked graphs* (i.e., graphs with interfaces) is defined in Section 2.

The algebraic structure of this category is axiomatized in Section 3 by the notion of *dgs-monoidality*: It is shown that the category of ranked graphs is isomorphic to the free dgs-monoidal category generated from a single arrow.

Lifting this structure of graphs to transformations in Section 4, they immediately get a distributed flavor, that is, transformations of complex graphs are build from transformations of simpler components. In fact, this notion of rewriting is closely related to distributed graph transformation in the sense of [14]. Technically speaking, the transformations are given by the cells of a 2-category freely generated from basic cells which represent the rules of the system. Such 2-categorical models are well-known for term rewriting: See e.g. [28, 31]. More recently, they have been applied to term graph rewriting [7].

2 Graphs

This section introduces (ranked) graphs as isomorphism classes of (ranked) concrete graphs. This presentation departs slightly from the standard definition (see for example [13]), because our main concern is the algebraic structure of graphs.

Definition 1 (directed concrete graphs). A *(directed) concrete graph* d is a four-tuple $d = \langle E, N, s, t \rangle$, where E is a set of *edges*, N is a set of *nodes*, $s, t : E \to N$ are functions called respectively the *source* and *target* function (and we shall often denote these components by $N(d)$, $E(d)$, s_d and t_d, respectively). A node n is *isolated* if it has neither ingoing nor outgoing arcs. A graph is *discrete* if E is empty (or, equivalently, all nodes are isolated).

Let d and d' be two concrete graphs. A *concrete graph morphism* $f : d \to d'$ is a pair of functions $\langle f_n, f_e \rangle$ such that $f_n : N(d) \to N(d')$ and $f_e : E(d) \to E(d')$. These functions must preserve source and target, i.e., for each edge $e \in E(d)$, $s_{d'}(f_e(e)) = f_n(s_d(e))$, and $t_{d'}(f_e(e)) = f_n(t_d(e))$.

Concrete graphs and their morphisms form a category we denote **DCG**. □

In the following, for each $i \in \mathbb{N}$ we denote by $\underline{i}$ the set $\{1, \ldots, i\}$ (thus $\underline{0} = \emptyset$).

Definition 2 (ranked concrete graphs and graphs). An (i,j)-*ranked concrete graph* (or also, a *concrete graph of rank* (i,j)) is a triple $g = \langle r, d, v \rangle$, where d is a concrete graph, $r : \underline{i} \to N(d)$ is a function called the *root mapping*, and $v : \underline{j} \to N(d)$ is a function called the *variable mapping*. Node $r(l)$ is called the l-*th root* of d, and $v(k)$ is called the k-*th variable* of d, for each admissible j, k.

Two (i,j)-ranked concrete graphs $g = \langle r, d, v \rangle$ and $g' = \langle r', d', v' \rangle$ are *isomorphic* if there exists a *ranked concrete graph isomorphism* $\phi : g \to g'$, i.e., a concrete graph isomorphism $\phi : d \to d'$ such that $\phi \circ r = r'$ and $\phi \circ v = v'$. A (i,j)-*ranked graph* G (or *with rank* (i,j)) is an isomorphism class of (i,j)-ranked concrete graphs. We shall often write G^i_j to recall that G has rank (i,j). □

The idea of equipping graphs with lists of distinguished nodes in order to define composition operations on them is not new (see for example [1, 16]): Roughly, for a graph G^i_j, the components $\underline{i}, \underline{j}$ represent *discrete interfaces*, through which graphs can be equipped with a compositional structure. And, dealing with isomorphism classes of concrete graphs, we disregard the concrete identity of nodes and edges when manipulating graphs.

We introduce now two operations on ranked graphs. The *composition* of two ranked graphs is obtained by gluing the variables of the first one with the roots of the second one, and it is defined only if their number is equal. This operation allows us to define a category having ranked graphs as arrows. Next the *union* of graphs is introduced: It is always defined, and it is a sort of disjoint union where roots and variables are suitably renumbered. This second operation provides the category of graphs with a monoidal structure, made explicit in Section 3.3.

Definition 3 (the category of graphs). Let $G'^j_k = [\langle r', d', v' \rangle]$ and $G^i_j = [\langle r, d, v \rangle]$ be two ranked graphs. Their *composition* is the ranked graph $H^i_k = G'^j_k ; G^i_j$ defined as $H^i_k = [\langle in_d \circ r, d'', in_{d'} \circ v' \rangle]$, where $\langle d'', in_d, in_{d'} \rangle$ is a pushout of $\langle v : \underline{j} \to d, r' : \underline{j} \to d' \rangle$ in **DCG** (set $\underline{j}$ regarded as a discrete concrete graph).

DG denotes the category having as objects underlined natural numbers, and as arrows from $\underline{j}$ to $\underline{i}$ all (i,j)-ranked graphs. Arrow composition is defined as in Definition 3, and the identity on $\underline{i}$ is the graph G^i_{id} of rank (i,i) having i nodes, and where the k-th root is also the k-th variable, for all $k \in \underline{i}$. □

The well-definedness of graph composition easily follows from the uniqueness of pushouts, up-to isomorphism. Then it is easy to check that **DG** is a well-defined category, because composition is associative, and the identity laws hold.

Definition 4 (union of ranked graphs). Let $G_j^i = [\langle r, d, v\rangle]$ and $G'^k_l = [\langle r', d', v'\rangle]$ be two ranked graphs. Their *union* or *parallel composition* is the graph of rank $(i+k, j+l)$ $G_j^i \oplus G'^k_l = [\langle r'', d \uplus d', v''\rangle]$, where $\langle d \uplus d', p_0, p_1\rangle$ is the coproduct of d and d' in **DCG**, and $r'' : \underline{i+k} \to d \uplus d'$ and $v'' : \underline{j+l} \to d \uplus d'$ are the morphisms induced by the universal property. □

Example 1 (graphs, composition and union). Figure 1 shows four graphs. Nodes are graphically represented by a dot, "•", from (to) where the edges leave (arrive); the list of numbers on the left (right) represent pointers to the variables (roots): A dashed arrow from j to a node indicates that it is the j-th variable (root). For example, graph G_2 has rank $(1, 4)$, with five nodes, two of them isolated: There is only one "root node" (pointed by 1) with two outgoing edges to "variable nodes" (pointed by 1 and 4).

These graphical conventions make easy the operation of composition, that can be performed by matching the roots of the first graph with the variables of the second one, and then by eliminating them. For example, graph $G_1 \,;\, G_2$ is the composition of G_1 and G_2, of rank $(1, 2)$. The last graph is $G_1 \oplus G_2$, the union of G_1 and G_2, of rank $(5, 6)$. □

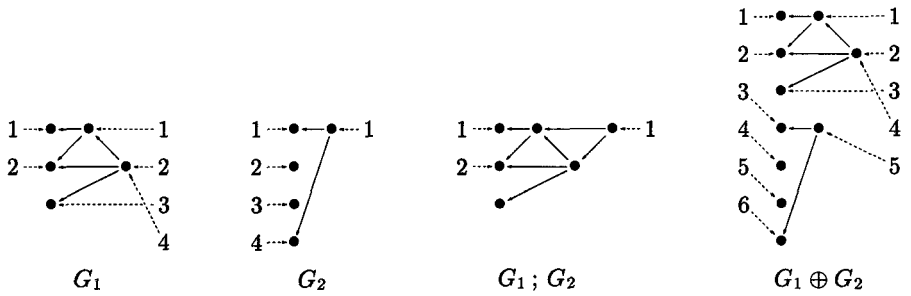


Fig. 1. Two graphs, their composition and their union.

We are now ready to show the main result of this section, namely, that every graph can be built from a small set of atomic ones.

Definition 5 (atomic graphs). An *atomic graph* is one of those graphs depicted in Figure 2. □

Theorem 6 (decomposition of graphs). *Every graph can be obtained as the value of an expression containing only atomic graphs as constants and composition and union as operators.* □

Space limitations force us to omit the proof. Nevertheless, a similar one for (acyclic) term graph can be actually found in [9], carried out by induction on

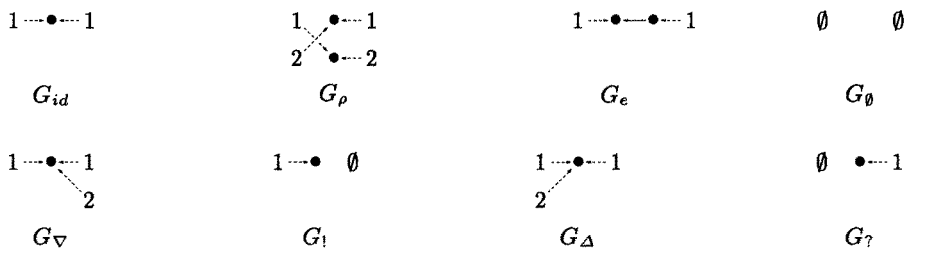
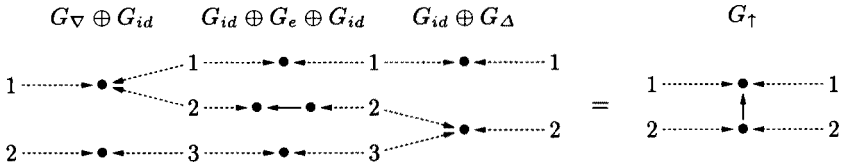


Fig. 2. Atomic graphs.

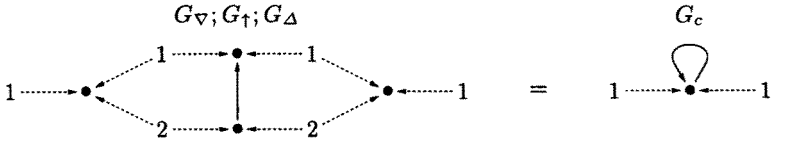
the number of nodes of a given graph. In this version, the main technical tool is the use of a suitable composition of G_∇ and G_Δ to induce cycles, as shown in Example 2: Such a remark is the basis of our finitary description of *traced monoidal categories*, to be found in next section.

Example 2 (a few graphs). In this example we want to use the operators for building a few simple graphs. The picture below shows the construction of the $(2,2)$-ranked graph $G_\uparrow$ as the value of the expression $G_\nabla \oplus G_{id}; G_{id} \oplus G_e \oplus G_{id}; G_{id} \oplus G_\Delta$ (with "$\oplus$" binding stronger than ";"). Here, G_{id} is used to create isolated nodes which are "visible" (so to say) from both interfaces, G_e provides the edge, and G_∇ and G_Δ are used to glue the source and the target of this edge with the two additional vertices:



The "downward" edge between two visible nodes is obtained by applying permutations, like in $G_\rho; G_\uparrow; G_\rho$.

Loops are created by gluing together vertices, that is, $G_c = G_\nabla; G_\uparrow; G_\Delta$ is the graph of rank $(1,1)$:



In order to restrict e.g. the right interface, we may compose with $G_!$: Then $G_c; G_!$ yields the $(0,1)$-ranked loop. □

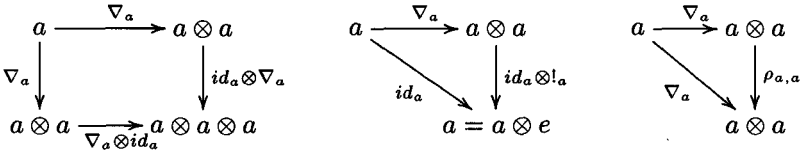
3 Graphs as Terms

The aim of this section is to present a complete axiomatization for the category **DG** of ranked graphs, as done for (cyclic) term graphs in [9]. We first introduce the notion of *dgs-monoidal categories*; we then show a finitary encoding of *traced monoidal categories* into the dgs-monoidal structures, from which the completeness result can be inferred via a folklore characterization of hyper-graphs [33, 21].

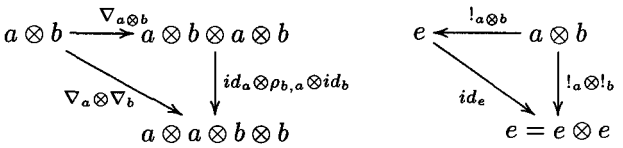
3.1 On (d)gs-monoidal categories

In this section we introduce *dgs-monoidal* categories, an extension of *gs-monoidal* ones [9, 7], which are used for our equational presentation of graphs.

Definition 7 (gs-monoidal categories). A *gs-monoidal category* $\mathbf{C}$ is a six-tuple $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, ! \rangle$, where $\langle \mathbf{C}_0, \otimes, e, \rho \rangle$ is a symmetric strict monoidal category and $! : Id \Rightarrow e : \mathbf{C}_0 \to \mathbf{C}_0$, $\nabla : Id \Rightarrow \otimes \circ D : \mathbf{C}_0 \to \mathbf{C}_0$ are two transformations (D is the diagonal functor), such that $!_e = \nabla_e = id_e$ and satisfying the *coherence* axioms



and the *monoidality* axioms



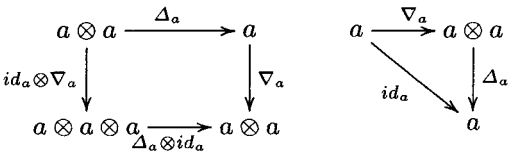
A *gs-monoidal functor* $F : \mathbf{C} \to \mathbf{C}'$ is a symmetric monoidal functor such that $F(!_a) = !'_{F(a)}$ and $F(\nabla_a) = \nabla'_{F(a)}$. The category of small gs-monoidal categories and their functors is denoted by **GS-Cat**. □

Introduced in [9], *gs-monoidal categories* can be roughly considered as monoidal categories, enriched by two transformations that allows for a controlled way of duplicating and discharging data. In fact, if we consider a single arrow $t : a \to b$ as a local state with interfaces a and b being pointers to internal data structures, then $!_b : b \to e$ can be considered as a *discharger* of the pointer b, so that $t; !_b$ represents the same structure of t, but with an empty interface. Similarly, $\nabla_b : b \to b \otimes b$ represents a *duplication* of the pointer b, so that $t; \nabla_b$ can be seen as a *shared* instance of t.

These structures fill the gap between monoidal and cartesian categories: It can be considered categorical folklore [18, 25, 19, 7] that, equipping a monoidal category with suitable *natural* transformations, we obtain a cartesian category: See e.g. [9] for a recollection. In our case, an instance of the theorem is obtained

simply requiring the naturality of ∇ and !: Such naturality forces a correspondence between pointers and underlying structures (so that e.g. $t;!_b = !_a$: deleting a pointer is the same as deleting the structure), and allows to recast the usual notion of *algebraic theory* [26].

Definition 8 (dgs-monoidal categories). A *dgs-monoidal category* $\mathbf{C}$ is a eight-tuple $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, !, \Delta, ? \rangle$, such that both the six-tuples $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, ! \rangle$ and $\langle (\mathbf{C}_0)^{op}, \otimes, e, \rho, \Delta, ? \rangle$ are gs-monoidal categories (where $(\mathbf{C}_0)^{op}$ is the dual category of $\mathbf{C}_0$), and satisfying



A *dgs-monoidal functor* $F : \mathbf{C} \to \mathbf{C}'$ is a gs-monoidal functor such that also F^{op} is gs-monoidal. The category of small dgs-monoidal categories and their functors is denoted by **DGS-Cat**. □

Equivalent notions of dgs-monoidal categories have surfaced quite frequently in recent years. A (bicategorical) presentation is used as a description of the (bi)category of relations already in [5], which forms the basis for some recent work on the categorical description of *circuits* [21, 17]: Arrows are processes, and the bicategorical structure allows to relate specifications which are equivalent but show a different internal structure (a different *implementation*, so to say).

Intuitively, the arrow $?_a$ can be interpreted as the NEW operator, creating a pointer to a new name of sort a; while Δ_a corresponds to a *matching*, forcing the equality of two pointers. Some of the additional axioms can then be explained as *housekeeping* operations on the set of pointers of a structure. For example, the axiom $\nabla_a; \Delta_a$ simply states that duplicating the pointers to a structure, and then equating them, results in an unchanged set of pointers. Instead, since $?_a$ creates a new name, and $!_a$ deletes only the associated pointer, *without* destroying the new name, then $?_a; !_a$ is in general different from id_e.

3.2 On the notion of feedback

Traced monoidal categories have been studied *per se* as a categorical tool [20]. However, they already surfaced in the literature related to algebraic theories. In fact, there is a strong connection between traced categories and *iteration theories* (that is, algebraic theories with an equational characterization of (least) fix-point [2, 3]) as it is pointed out in the works on *flownomial calculus* [32, 33].

Definition 9 (traced monoidal categories). A *traced monoidal category* $\mathbf{C}$ is a five-tuple $\langle \mathbf{C}_0, \otimes, e, \rho, tr \rangle$, where $\langle \mathbf{C}_0, \otimes, e, \rho \rangle$ is a symmetric strict monoidal

category,[3] which is equipped with a family of functions $tr^u_{a,b} : \mathbf{C}[a \otimes u, b \otimes u] \to \mathbf{C}[a, b]$, satisfying the *naturality* axioms

$$tr^u_{a,b}(f; (id_b \otimes g)) = tr^u_{a,b}((id_a \otimes g); f) \qquad tr^u_{c,d}((h \otimes id_u); f; (l \otimes id_u)) = h; tr^u_{a,b}(f); l$$

the *vanishing* axioms

$$tr^e_{a,b}(f) = f \qquad tr^v_{a,b}(tr^u_{a \otimes v, b \otimes v}(f)) = tr^{u \otimes v}_{a,b}(f)$$

the *superposing* axiom

$$tr^u_{a \otimes c, b \otimes d}((id_a \otimes \rho_{c,u}); (f \otimes g); (id_b \otimes \rho_{u,d})) = tr^u_{a,b}(f) \otimes g$$

and the *yanking* axiom $tr^u_{u,u}(\rho_{u,u}) = id_u$.

A *traced gs-monoidal category* $\mathbf{C}$ is a seven-tuple $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, !, tr \rangle$, such that $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, ! \rangle$ is a gs-monoidal category and $\langle \mathbf{C}_0, \otimes, e, \rho, tr \rangle$ is a traced monoidal one. A *traced dgs-monoidal category* $\mathbf{C}$ is a nine-tuple $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, !, \Delta, ?, tr \rangle$ such that $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, !, \Delta, ? \rangle$ is a dgs-monoidal category, $\langle (\mathbf{C}_0)^{op}, \otimes, e, \rho, tr \rangle$ is a traced monoidal one, and satisfying

$$tr^u_{e,u}(\nabla_u) = ?_u \qquad tr^u_{u,e}(\Delta_u) = !_u$$

A *traced monoidal* (*gs-monoidal, dgs monoidal*) functor is a symmetric monoidal (gs-monoidal, dgs-monoidal) functor such that $F(tr^u_{a,b}(t)) = tr^{F(u)}_{F(a),F(b)}(F(t))$. The category of small traced monoidal (gs-monoidal, dgs-monoidal) categories and their functors is denoted by **Tr-Cat** (**TrGS-Cat** and **TrDGS-Cat**). □

A correspondence theorem is proved in [9], namely the existence of an isomorphism between the category of term graphs over a signature Σ and the free traced gs-monoidal category over Σ. As suggested in [21], the dgs-monoidal is the proper one to deal instead with *term hyper-graphs*: In fact, they prove a correspondence theorem between (a logical presentation of) term hyper-graphs (called circuits) over a given *hyper-signature* and (an equivalent version of) traced dgs-monoidal categories. The encoding result proved in the next section (see Theorem 11) will allow us to state our completeness result as a corollary of this property.

3.3 A finite description for traces

We open the section showing that **DG** can be equipped with a dgs-monoidal structure. We then prove the encoding of a traced category into the dgs-monoidal one, deriving this way our completeness result.

Proposition 10 (DG is dgs-monoidal). *The category* **DG** *of ranked graphs has a dgs-monoidal structure, given by the eight-tuple*

$$\langle \mathbf{DG}, \oplus, G_{id}, G_\rho, G_\nabla, G_!, G_\Delta, G_? \rangle$$

where the auxiliary arrows are (defined as) the atomic graphs of Definition 5. □

[3] When introduced in [20], the authors dealt with the more general *balanced monoidal categories* where, basically, the family of symmetries is not unique.

The next theorem shows that dgs-monoidal categories are also traced, via a simple encoding for traces which seems not to be known in the literature.[4]

Theorem 11 (encoding feedback). *Let* $\mathbf{C}$ *be a dgs-monoidal category* $\langle \mathbf{C}_0, \otimes, e, \rho, \nabla, !, \Delta, ? \rangle$. *Then it admits also a traced dgs-monoidal structure: The family "tr" is defined as*

$$tr^u_{a,b}(f) = (id_a \otimes \nabla^h_u); (f \otimes id_u); (id_a \otimes \Delta^h_u)$$

where ∇^h_u, Δ^h_u *denote* $?_u; \nabla_u$ *and* $\Delta_u; !_u$, *respectively.*

Proof. All the additional axioms for traced dgs-monoidal categories are easy to check (by directly substituting the derived operators, via the axioms for dgs-monoidality) except for the first naturality axioms. As an example, the superposing axiom can be simply checked as

$$\begin{array}{ll} tr^u_{a\otimes c, b\otimes d}((id_a \otimes \rho_{c,u}); (f \otimes g); (id_b \otimes \rho_{u,d})) & = \\ (id_a \otimes id_c \otimes \nabla^h_u); [(id_a \otimes \rho_{c,u}); (f \otimes g); (id_b \otimes \rho_{u,d}) \otimes id_u]; (id_b \otimes id_d \otimes \Delta^h_u) & = \\ (id_a \otimes \nabla^h_u); f; (id_b \otimes \Delta^h_u) \otimes g & = \\ tr^u_{a,b}(f) \otimes g & \end{array}$$

via the naturality of ρ, and the axioms $\rho_{e,c} = \rho_{e,d} = id_e$.

Let us move now to the first naturality axiom. We need to use a decomposition property, suggested in [21]: For each $f : a \to c$ and $g : c \to b$, the equality $f; g = (id_a \otimes \nabla^h_c); (f \otimes id_c \otimes g); (\Delta^h_c \otimes id_b) = (\nabla^h_c \otimes id_a); (g \otimes id_c \otimes f); (id_b \otimes \Delta^h_c)$ holds. Then, the proof goes in the following way

$$\begin{array}{ll} tr^u_{a,b}(f; (id_b \otimes g)) & = \\ (id_a \otimes \nabla^h_u); [f; (id_b \otimes g) \otimes id_u]; (id_b \otimes \Delta^h_u) & = \\ \nabla^*_{a\otimes u}; [f; (id_b \otimes g) \otimes !_a \otimes ?_b \otimes id_u]; \Delta^*_{b\otimes u} & = \\ \nabla^*_{a\otimes u}; [(id_a \otimes id_u \otimes \nabla^h_{b\otimes u}); (f \otimes id_b \otimes id_u \otimes g \otimes id_u); & \\ (\Delta^h_{b\otimes u} \otimes id_b \otimes id_u) \otimes !_a \otimes ?_b \otimes id_u]; \Delta^*_{b\otimes u} & = \\ \nabla^*_{a\otimes u}; [!_a \otimes ?_b \otimes u \otimes (\nabla^h_{b\otimes u} \otimes id_a \otimes id_u); (g \otimes id_u \otimes id_b \otimes id_u \otimes f); & \\ (id_b \otimes id_u \otimes \Delta^h_{b\otimes u})]; \Delta^*_{b\otimes u} & = \\ \nabla^*_{a\otimes u}; [!_a \otimes ?_b \otimes u \otimes (g \otimes id_u); f]; \Delta^*_{b\otimes u} & = \\ \nabla^*_{a\otimes u}; [(id_a \otimes g); f \otimes !_a \otimes ?_b \otimes id_u]; \Delta^*_{b\otimes u} & = \\ (id_a \otimes \nabla^h_u); [(id_b \otimes g); f \otimes id_u]; (id_b \otimes \Delta^h_u) & = \\ tr^u_{a,b}((id_a \otimes g); f) & \end{array}$$

where $\nabla^*_{a\otimes u}$, $\Delta^*_{b\otimes u}$ denote $(id_a \otimes ?_u); \nabla_{a\otimes u}$ and $\Delta_{b\otimes u}; (id_b \otimes !_u)$, respectively. □

We are finally able to provide our main theorem for the section: It relies on the correspondence results given for traced dgs-monoidal theories in the literature, coupled with the properties we proved with Proposition 10 and Theorem 11.

Theorem 12 (DG is a free structure). *Let* d *be a concrete graph with just one node and one arrow, and let* $\mathbf{DGS}(d)$ *be the associated free dgs-monoidal category. Then* $\mathbf{DGS}(d)$ *is isomorphic to* $\mathbf{DG}$ *via a dgs-monoidal functor.* □

[4] We just discovered an equivalent description, provided without proof, in [22].

4 A 2-category for Graph Rewriting

We open this section recalling a few definitions about the *double-pushout* approach to graph transformation. We will then present the basic notions regarding *2-categories*: They will be used to provide an alternative presentation of graph transformation, which we will prove equivalent to the traditional one.

4.1 Basic notions of double-pushout

Historically, the first of the algebraic approaches to graph transformation is the so-called *double-pushout (DPO) approach* introduced in [15], which owes its name to the basic construction used to define a single derivation step, modeled indeed by two gluing diagrams (i.e., pushouts) in the category **DCG** of concrete graphs.

Definition 13 (graph productions). A *graph production* $p : s$ is composed of a *production name* p and of a span of injective graph morphisms $s = \langle L \leftarrow_l K \rightarrow_r R \rangle$, called *production span.* A *graph transformation system* $\mathcal{G}$ is a set of graph productions (with different names). □

Derivation steps in the DPO approach are defined operationally, as double pushout constructions: The left-hand side L contains the items that must be present for an application of the production, the right-hand side R those that are present afterwards, and the context graph K specifies the "gluing items", i.e., the objects which are read during application but are not consumed.

Definition 14 (DPO derivation). A *double-pushout* is a diagram like in Figure 3, where top and bottom are production spans and (1) and (2) are pushouts. If $p : \langle L \leftarrow_l K \rightarrow_r R \rangle$ is a production, a *derivation step* from G to H via production p and "context embedding" d is denoted by $G \Rightarrow_{p/d} H$.

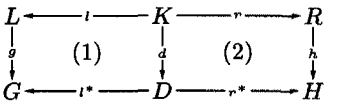


Fig. 3. A DPO derivation step

More abstractly, we denote by $\Rightarrow_{\mathcal{G}} \subseteq |\mathbf{DCG}| \times |\mathbf{DCG}|$ the corresponding *rewrite relation* on graphs, and by $\Rightarrow^*_{\mathcal{G}}$ its reflexive and transitive closure. □

The existence of a derivation is characterized by the gluing conditions, which characterizes the existence (and uniqueness up-to isomorphism) of the *pushout complement*, i.e., the context graph D and morphisms l^* and d such that subdiagram (1) in the left of Figure 3 is a pushout. Operationally speaking, the application of a production $p : \langle L \leftarrow_l K \rightarrow_r R \rangle$ to a graph G consists of three steps. First, the match $g : L \rightarrow G$ has to be chosen, providing an occurrence

of L in G, such that the gluing conditions are satisfied. Then, all objects of G matched by $L - l(K)$ are removed. This leads to the context graph D. Finally, the objects of $R - r(K)$ are added to D, to obtain the derived graph H.

4.2 A few notions on 2-categories

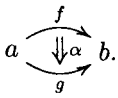
In this section we briefly present some notions about 2-categories needed in the rest of the paper. We first recall the basic definitions: For an introduction, we refer the reader to the classical work [23].

Definition 15 (2-categories). A *2-category* $\underline{\mathbf{C}}$ is a four-tuple $\langle Ob_c, \mathbf{C}, *, id\rangle$ such that Ob_c is a set of *2-objects* and, indexed by elements in Ob_c, $\mathbf{C}$ is a family of categories $\mathbf{C}[a,b]$ (the *hom-categories* of $\underline{\mathbf{C}}$), $*$ is a family of functors $*^b_{a,c} : \mathbf{C}[a,b] \times \mathbf{C}[b,c] \to \mathbf{C}[a,c]$ and id is a family of objects $id_a \in |\mathbf{C}[a,a]|$, satisfying for each $\alpha \in \mathbf{C}[a,b]$

$$[\textbf{category}] \qquad id_a * \alpha = \alpha = \alpha * id_b \qquad (\alpha * \beta) * \gamma = \alpha * (\beta * \gamma)$$

where for the sake of readability the indexes of $*$ are dropped, and the arrow id_{id_a} (the identity on the object id_a) is denoted by the object itself. □

The *category* axioms impose a categorical structure to the pair $\mathbf{C}_u = \langle Ob_c, |\mathbf{C}[a,b]|\rangle$ (denoted as the *underlying category* of $\underline{\mathbf{C}}$), where the objects are the elements of the set of 2-objects, and the elements of the hom-set $\mathbf{C}_u[a,b]$ are the objects of the hom-categories $\mathbf{C}[a,b]$. Then, roughly, a 2-category can be simply described as a category $\mathbf{C}$ such that, given any two objects a, b, the hom-set $\mathbf{C}[a,b]$ is actually a category. In fact, we denote as *arrows* and *cells* of the 2-category $\underline{\mathbf{C}}$ the objects and arrows of the hom-categories, respectively: By $\alpha : f \Rightarrow g : a \to b$ we mean that α is a cell in $\mathbf{C}[a,b]$ from f to g, depicted as



Definition 16 (2-functors). Let $\underline{\mathbf{C}}$, $\underline{\mathbf{D}}$ be 2-categories. A *2-functor* $F : \underline{\mathbf{C}} \to \underline{\mathbf{D}}$ is a pair $\langle F_o, F_m\rangle$, where $F_o : Ob_c \to Ob_d$ is a function and F_m is a family of functors $F_{a,b} : \mathbf{C}[a,b] \to \mathbf{C}[F_o(a), F_o(b)]$.

We denote **2Cat** the category of (small) 2-categories and 2-functors. □

The paradigmatic example of 2-category is $\underline{\mathbf{Cat}}$, having small categories as objects, functors as arrows, and natural transformations as cells. As far as rewriting is concerned, it is well-known that a suitable class of 2-categories, *algebraic 2-theories*, can be used for describing term rewriting (see e.g. [31, 28], and the more recent [34, 11]), while in [7, 8] similar categorical models have been proposed for term graph rewriting, *(traced) gs-monoidal 2-theories.*

A *computad* [35] is a category equipped with a graph structure over hom-sets (informally, a set of cells not closed under composition), which intuitively

represent a rule based system, the states being the arrows and the rules the cells. The main fact for our analysis is that from a computad a (structured) 2-category is generated in a free way, by closing the cells under all relevant operations.

Definition 17 (computads). A *computad* $\mathbf{C}_S$ is a pair $\langle \mathbf{C}, S \rangle$, where $\mathbf{C}$ is a category and S is a set of "cells" over the hom-sets of $\mathbf{C}$. A *computad morphism* $\langle F, h \rangle : \mathbf{C}_S \to \mathbf{D}_T$ is a pair such that $F : \mathbf{C} \to \mathbf{D}$ is a functor and $h : S \to T$ is a function, preserving source and target of the cells in the expected way.

Computads and their morphisms form a category, denoted **Comp**. □

Proposition 18 (free 2-categories). *Let $V_2 : \mathbf{2Cat} \to \mathbf{Comp}$ be the forgetful functor mapping a 2-category to the underlying computad (simply forgetting cell composition): It admits a left adjoint $F_2 : \mathbf{Comp} \to \mathbf{2Cat}$.* □

Intuitively, the free functor F_2 composes the cells of a computad in all the possible ways, both horizontally and vertically, imposing further equalities in order to satisfy the axioms of a 2-category.

4.3 From DPO derivations to cells

In this section we show how from a graph transformation system we can obtain a suitable computad, such that the cells of the freely generated 2-category faithfully describe the derivations of the system.

Definition 19 (discrete graph production). A graph production $p : \langle L \leftarrow_l K \rightarrow_r R \rangle$ is *discrete* if K is discrete. A graph transformation system is discrete if all its productions are so. □

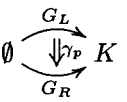
Given a production $p : \langle L \leftarrow_l K \rightarrow_r R \rangle$, the associated *discrete production* is obtained taking the discrete graph K_d underlying the context graph K. It is well known that, from the point of view of the rewrite relation, there is no difference between a graph transformation system and the associated discrete instance.

Definition 20 (induced discrete productions). Given a production $p : \langle L \leftarrow_l K \rightarrow_r R \rangle$, we denote as γ_p the associated *discrete production*, obtained by taking the discrete graph K_d underlying the context graph K and restricting l and r. Given a graph transformation system $\mathcal{G}$, we denote by $\mathcal{G}_d$ the corresponding discrete one obtained by "making discrete" the productions of $\mathcal{G}$. □

Proposition 21 (discrete rewrites). *Let $\mathcal{G}$ be a graph transformation system: Then, $\Rightarrow_{\mathcal{G}} = \Rightarrow_{\mathcal{G}_d}$.* □

From now on we shall assume, without loss of generality, that for each discrete graph K, its set of nodes is formed by natural numbers, so that $\underline{j} = N(K)$ for $j \in \mathbb{N}$. We can now provide the translation from a discrete graph transformation system to a computad.

Definition 22 (discrete computad). Let $\mathcal{G}$ be a discrete graph transformation system. The associated computad $\mathbf{C}_\mathcal{G}$ has **DG** as underlying category, and a cell γ_p for each production $p : \langle L \leftarrow_l K \rightarrow_r R \rangle \in \mathcal{G}$, such that

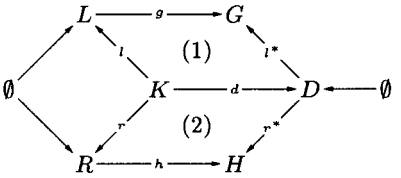


where K is regarded as the set $N(K)$, and G_L and G_R denote the ranked graphs $[\langle l, L, \emptyset \rangle]$ and $[\langle r, R, \emptyset \rangle]$. □

Then, we can finally prove our main theorem.

Theorem 23 (derivation as cells). *Let $\mathcal{G}$ be a graph transformation system. Then there exists a derivation $G \Rightarrow^*_\mathcal{G} H$ iff there exists a 2-cell from $[\langle \emptyset, G, \emptyset \rangle]$ to $[\langle \emptyset, H, \emptyset \rangle]$ in $F_2(\mathbf{C}_\mathcal{G})[\emptyset, \emptyset]$.*

Proof. Only if. The proof goes via induction on the length of a derivation. It is enough to note that the DPO step of Figure 3 is modeled via the cell $\gamma_p * [\langle \emptyset, D, k \rangle]$, depicted as



If (outline). By structural induction, with induction base the set of cells γ_p associated to the productions, we prove that the rewrite relation is preserved by composition, both inside each hom-category and with respect to each component of the family "$*$" of composition functors. □

5 Further Works

In our opinion, future work lies in the field of *concurrent semantics* of graph transformation. In fact, despite the correspondence we got is faithful at the level of the rewrite relation, we cannot recast the notions of *parallel derivations* and *shift equivalence* by means of the 2-categorical structure. Any solution should take into account three different aspects of the problem. First, the dgs-monoidal structure should be lifted to the 2-categorical level, as for similar results with *iteration 2-theories* [10, 4], also obtaining a generalized version of Theorem 11. This would provide us with the syntax of parallel productions and derivations.

Second, the structure of the interfaces should be enriched, at least by "isolated" edges, in order to overcome the restriction to productions with discrete interface graphs. In fact, while "making discrete" a production does not mean

any harm for the rewrite relation (cf. Proposition 21), it reduces the set of possible parallel derivations.

Finally, the most delicate point is to ensure that the equivalence induced on cells by the coherence axioms of 2-categories resembles the usual *shift equivalence* on abstract graph derivations [6], like it happens for *permutation equivalence* in categorical models of term rewriting (see e.g. [11]). Here, particular attention should be paid to the isomorphism question between graphs: A problem that is at the basis of the *standard isomorphisms* solution in [6].

References

1. M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.
2. S. Bloom and Z. Ésik. *Iteration Theories.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1993.
3. S. Bloom and Z. Ésik. Solving polinomials fixed point equations. In *Mathematical Foundations of Computer Science*, volume 841 of *LNCS*, pages 52–67. Springer Verlag, 1994.
4. S.L. Bloom, Z. Ésik, A. Labella, and E.G. Manes. Iteration 2-theories. In *Proceedings AMAST'97*, 1997. To appear.
5. A. Carboni and R.F.C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49:11–32, 1987.
6. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract Graph Derivations in the Double-Pushout Approach. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 86–103. Springer Verlag, 1994.
7. A. Corradini and F. Gadducci. A 2-categorical presentation of term graph rewriting. In *Proceedings CTCS'97*, volume 1290 of *LNCS*. Springer Verlag, 1997.
8. A. Corradini and F. Gadducci. Rewriting cyclic structures. draft, 1997.
9. A. Corradini and F. Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Applied Categorical Structures*, 1998. To appear. Available at `http://www.di.unipi.it/~gadducci/papers/aptg.ps`.
10. A. Corradini and F. Gadducci. Rational term rewriting. In M. Nivat, editor, *Proceedings FoSSaCS'98*, LNCS. Springer Verlag, 1998. To appear.
11. A. Corradini, F. Gadducci, and U. Montanari. Relating two categorical models of term rewriting. In *Rewriting Tecniques and Applications*, volume 914 of *LNCS*, pages 225–240. Springer Verlag, 1995.
12. A. Corradini and U. Montanari. An Algebra of Graphs and Graph Rewriting. In *Proceedings of the 4^{th} Summer Conference on Category Theory and Computer Science (CTCS '91)*, volume 530 of *LNCS*, pages 236–260. Springer Verlag, 1991.
13. 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.* World Scientific, 1997.
14. H. Ehrig, P. Boehm, U. Hummert, and M. Löwe. Distributed parallelism of graph transformation. In *13th Int. Workshop on Graph Theoretic Concepts in Computer Science*, volume 314 of *LNCS*, pages 1–19. Springer Verlag, 1988.

15. H. Ehrig, M. Pfender, and H.J. Schneider. Graph-grammars: an algebraic approach. In *Proceedings IEEE Conf. on Automata and Switching Theory*, pages 167–180, 1973.
16. A. Habel. *Hyperedge replacement: Grammars and languages*, volume 643 of *LNCS*. Springer Verlag, 1992.
17. U. Hensel and D. Spooner. A view on implementing processes: Categories of circuits. In M. Haveraaen, O. Owe, and O. Dahl, editors, *Recent Trends in Data Types Specification*, volume 1130 of *LNCS*, pages 237–255. Springer Verlag, 1995.
18. H.-J. Hoenke. On partial recursive definitions and programs. In *International Conference on Fundamentals of Computations Theory*, volume 56 of *LNCS*, pages 260–274. Springer Verlag, 1977.
19. B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69:73–106, 1994.
20. A. Joyal, R. Street, and D. Verity. Traced Monoidal Category. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:425–446, 1996.
21. P. Katis, N. Sabadini, and R.F.C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115:141–178, 1997.
22. P. Katis, N. Sabadini, and R.F.C. Walters. Span(graph): A categorical algebra of transition systems. to appear Proceedings AMAST'97, 1997.
23. G.M. Kelly and R.H. Street. Review of the elements of 2-categories. In G.M. Kelly, editor, *Proceedings of the Sydney Category Seminar*, volume 420 of *Lecture Notes in Mathematics*, pages 75–103. Springer Verlag, 1974.
24. J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 1–116. Oxford University Press, 1992.
25. Y. Lafont. Equational reasoning with 2-dimensional diagrams. In *Term Rewriting, French Spring School of Theoretical Computer Science*, volume 909 of *LNCS*, pages 170–195. Springer Verlag, 1995.
26. F.W. Lawvere. Functorial semantics of algebraic theories. *Proc. National Academy of Science*, 50:869–872, 1963.
27. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
28. A.J. Power. An abstract formulation for rewrite systems. In *Proceedings Category Theory in Computer Science*, volume 389 of *LNCS*, pages 300–312. Springer Verlag, 1989.
29. J. Power. A 2-categorical parsing theorem. *Journal of Algebra*, 129:439–445, 1990.
30. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific, 1997.
31. D.E. Rydehard and E.G. Stell. Foundations of equational deductions: A categorical treatment of equational proofs and unification algorithms. In *Proceedings Category Theory in Computer Science*, volume 283 of *LNCS*, pages 114–139. Springer Verlag, 1987.
32. G. Stefanescu. On flowchart theories: Part II. The nondeterministic case. *Theoret. Comput. Sci.*, 52:307–340, 1987.
33. G. Stefanescu. Algebra of flownomials. Technical Report SFB-Bericht 342/16/94 A, Technical University of München, Institut für Informatik, 1994.
34. J.G. Stell. *Categorical Aspects of Unification and Rewriting*. PhD thesis, University of Manchester, 1992.
35. R. Street. Categorical structures. In M. Hazewinkel, editor, *Handbook Of Algebra*, pages 529–577. Elsevier, 1996.