# Linear Ordered Graph Grammars and Their Algebraic Foundations *

Ugo Montanari[1] and Leila Ribeiro[2]

[1] Dipartimento di Informática, Universitá di Pisa,
Pisa, Italy
ugo@di.unipi.it,
[2] Instituto de Informática, Universidade Federal do Rio Grande do Sul,
Porto Alegre, Brazil
leila@inf.ufrgs.br

**Abstract.** In this paper we define a special kind of graph grammars, called linear ordered graph grammars, that can be used to describe distributed systems with mobility and object-based systems. Then we show how to model such grammars and their semantics in terms of tiles making explicit the aspects of interactivity and compositionality, that are of great importance in distributed systems.

## 1 Introduction

Wide area network programming is important and difficult to cope with using ad hoc techniques. In particular, testing in such environments has shown to be very inefficient due to the high dynamicity and possibility of failures of the environment. Thus, formal techniques to assure the correct behavior of the system are even more needed in this framework.

Graph transformation systems [18, 7] have been used in a wide variety of applications, and can provide suggestive and technically adequate models of computation, semantic foundations and verification methods. There have been many approaches to model distributed systems using graph transformation systems [19]. Considering that our aim is to be able to model mobility of processes within a system, a natural way of representing distributed systems graphically is to model communication channels as nodes (abstracting out the physical network), agents/processes/ambients as (hyper)arcs and ports as tentacles (for example, the modeling of $\pi$-calculus agents using graph transformations defined in [16]). If we want to have a more object-oriented view, we could see objects as nodes and messages as (hyper)arcs, as it was done in [6]. What is important to notice is that in both modelings, mobility involves the creation and passing of nodes, while the process/message interpretation of arcs requires resource consciousness ("linearity" in Girard sense). In the graph transformation modelings discussed above, this means that neither rules nor matches for rules may identify arcs (because they model resources). Actually, if we do not have this linearity,

* This work was partially supported by the projects AGILE (FET project), COMETA (MIUR project), IQ-Mobile (CNPq and CNR) and ForMOS (CNPq and Fapergs).

or resource consciousness, property, the construction of a suitable compositional true concurrency semantics (based on concatenable processes, for example) is not possible (this will be further explained in Sect. 4).

In this paper we propose to specialize the existing single pushout (SPO) approach to graph transformations [14, 8] to a kind of grammars that can be used to model both distributed systems with mobility and object-based systems. These grammars will be called *linear ordered graph grammars*. The linearity property is achieved by ordering nodes and arcs, and allowing only injective morphisms on items that represent resources (arcs).

Interactivity and abstract semantics via observations are key features of communicating systems. Compositionality is also very important for large distributed systems. Tiles have these as their main aims, and previous work has shown their feasibility [10, 9]. In addition, they exhibit an algebraic flavor (tiles have a straightforward axiomatization as monoidal double categories) which may allow for universal constructions, compositionality and refinement in the classical style of algebraic semantics [15]. We show that the representation of SPO graph transformations of linear ordered graph grammars within tile logic is simple and natural. In this way, we provide a basis for an algebraic theory of graph grammars where observations and interactions are in the foreground, and this is not usually emphasized in the DPO/SPO literature, although it is a very important aspect of the kind of systems we are interested in modeling

This paper is organized as follows: Sect. 2 reviews the basic concepts of SPO graph rewriting using typed hypergraphs, Sect. 3 presents the main concepts of tile systems, introducing the kind of arrows needed for modeling graphs and (partial) graph morphisms as tiles, Sect. 4 introduces linear ordered graph grammars and shows how their syntax and semantics can be represented as tiles/tile rewrite systems. Section 5 brings a summary of our main results.

## 2 Graph Rewriting

In this section we will introduce the basic concepts of graph rewriting in the single pushout (SPO) approach [14, 8]. However, instead of the standard definitions of morphisms based on total morphisms from subgraphs, we will explicitly use partial functions and a weak commutativity requirement (instead of the usual commutativity). In [13] it was shown that the two definitions are equivalent. Moreover, the definition based on partial functions has two advantages: it allows us to define typed hypergraph categories in an easy way (as instances of generalized graph structure categories), and it makes clearer the correspondence to the tile rewriting model that will be defined in Sect. 4.

**Definition 1 (Weak Commutativity).** *For a (partial) function $f : A \to B$ with domain $dom(f)$, let $f? : A \leftarrow dom(f)$ and $f! : dom(f) \to B$ denote the corresponding domain inclusion and the domain restriction. Given functions $a : A \to A'$, $b : B \to B'$ and $f' : A' \to B'$, where $a$ and $b$ are total, we write $f' \circ a \geq b \circ f$ and say that the diagram* ***commutes weakly*** *iff $f' \circ a \circ f? = b \circ f!$.*

*Remark 1.* If $f$ and $f'$ are total, weak commutativity coincides with commutativity. Note that $((f?)^{-1}, f!)$ is a ***factorization*** of $f$.

The compatibility condition defined above means that everything that is mapped by the function must be compatible. The term "weak" is used because the compatibility is just required on preserved (mapped) items, not on all items.

Now we will define hypergraphs and partial morphisms. In this paper we will use undirected hypergraphs, and therefore we will only define a *source* function connecting each arc with the list of nodes it is connected to. We use the notation $S^*$ to denote the set of all (finite) lists of elements of $S$.

**Definition 2 ((Hyper)Graph, (Hyper)Graph Morphism).** *A **(hyper) graph** $G = (N_G, A_G, source^G)$ consists of a set of nodes $N_G$, a set of arcs $A_G$ and a total function $source^G : A_G \to N_G^*$, assigning to each arc a list of nodes.*

*A **(partial) graph morphism** $g : G \to H$ from a graph $G$ to a graph $H$ is a pair of partial functions $g_N : N_G \to N_H$ and $g_A : A_G \to A_H$ which are weakly homomorphic, i.e., $g_N^* \circ source^G \geq source^H \circ g_A$ ($g_N^*$ is the extension of $g_N$ to lists of nodes). A morphism is called total if both components are total. The category of hypergraphs and partial hypergraph morphisms is denoted by* **HGraphP** *(identities and composition are defined componentwise).*

To distinguish different kinds of nodes and arcs, we will use a notion of typed hypergraphs, analogous to typed graphs [5, 12][3].

**Definition 3 (Typed Hypergraphs).** *A typed hypergraph is a tuple $HG^{TG} = (HG, type^{HG}, TG)$ where $HG$ and $TG$ are hypergraphs, called instance and type graph, respectively, and $type^{HG} : HG \to THG$ is a total hypergraph morphism, called typing morphism.*

*A morphism between typed hypergraphs $HG1^{TG}$ and $HG2^{TG}$ is a partial function $f : HG1 \to HG2$ such that $type1^{HG1} \geq type2^{HG2} \circ f$. The category of all hypergraphs typed over a type graph $TG$, denoted by* **THGraphP(TG)**, *consists of all hypergraphs over $TG$ as objects and all morphisms between typed hypergraphs (identities and composition are the identities and composition of partial functions).*

For the well-definedness of the categories above and the proof that these categories have pushouts we refer to [13] (these categories can be constructed as generalized graph structures categories). The construction of pushouts in such categories is done componentwise, followed by a free construction to make the *source* and *type* components total (actually, as types are never changed by morphisms, this free construction is not needed in the case of typed graphs). In the rest of the paper, we will usually denote typed hypergraphs just by graphs.

**Definition 4 (Graph Grammar).** *Given a (hyper)graph $TG$, a **rule** is a morphism in* **THGraphP(TG)**. *A **graph grammar** is a tuple $GG = (TG, IG, Rules)$ where $TG$ is a (hyper)graph, called type graph, $IG$ is a (hyper)graph typed over $TG$, called initial or start graph, and $Rules$ is a set of rules over $TG$.*

[3] Note that, due to the use of partial morphisms, this is not just a comma category construction: the morphism *type* is total whereas morphisms among graphs are partial, and we need weak commutativity instead of commutativity. In [13] a way to construct such categories was defined.

**Definition 5 (Derivation Step, Sequential Semantics of a Graph Grammar).** *Given a graph grammar $GG = (TG, IG, Rules)$, a rule $r : L \to R \in Rules$, and a graph $G1$ typed over $TG$, a* ***match*** *$m : L \to G1$ for rule $r$ in $G1$ is a total morphism in* $\mathbf{THGraphP(TG)}$*. A* ***derivation step*** *$G1 \overset{r,m}{\Longrightarrow} G2$ using rule $r$ and match $m$ is a pushout in the category* $\mathbf{THGraphP(TG)}$*. The morphism $r'$ is called* ***co-rule****.*

$$\begin{array}{ccc} L & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle m'} \\ G1 & \xrightarrow[r']{} & G2 \end{array}$$

*A derivation sequence of $GG$ is a sequence of derivation steps $G_i \overset{r_i,m_i}{\Longrightarrow} G_{i+1}$, $i \in \{0..n\}$, $n \in \mathbb{N}$, where $G_0 = IG$, $r_i \in Rules$. The composition of all co-rules $r'_i$ is called* ***derived production*** *of a sequential derivation. The* ***sequential semantics*** *of $GG$ is the set of all sequential derivations of $GG$.*

An example of a rule and derivation using typed hypergraphs is shown in Figure 1. Arcs are drawn as squares and nodes as bullets. The tentacles of the arcs describe the *source* function and are numbered to indicate their order. Indices on nodes and arcs here just indicate different instances of items with same types. Intuitively, arcs represent resources, for example agents or ambient names, and nodes represent ambients. The rule may describe a kind of *open* operation of ambients and the creation of a new ambient: agent $B$ is preserved, the ambient represented by $\bullet_1$ (that is inside ambient $\bullet_2$ with name $A$) is opened, having the effect that all agents and ambients that were inside $\bullet_2$ before the application of the rule are now inside ambient $\bullet_1$, and a new ambient inside $\bullet_1$ is created (ambient $\bullet_2$ of $R$, named $C$). Note that this interpretation does not correspond exactly to a rule of the ambient calculus, since the agent that triggers the execution ($B$) is preserved, instead of being substituted by its continuation. This rule was chosen to illustrate using one example that our framework is expressive enough to model name fusion, a feature that is needed in some calculi (for example, in the fusion calculus by Parrow and Victor, as well as in the ambient calculus) and preservation of resources, needed to model, for example, replication (! operator in the *pi*-calculus).

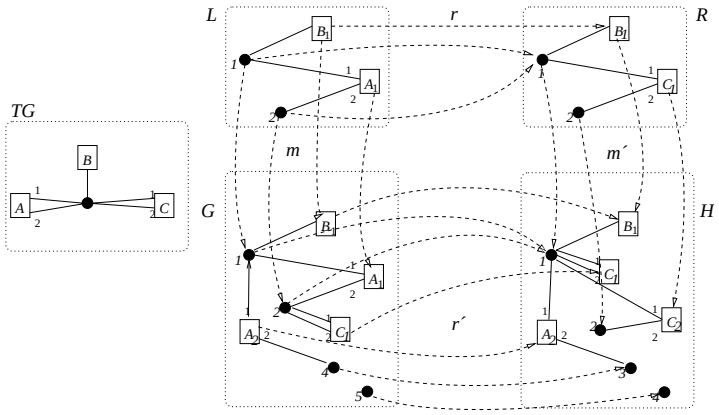


**Fig. 1.** Derivation using rule $r$ at match $m$

## 3 Tiles and Algebraic Semantics

In this section we will define the basic notions of tile rewrite systems. Each tile can be seen as a square consisting of four arrows: the horizontal ones describe states, and the vertical ones observations. First we define some monoidal theories that will be used to build suitable arrows of tiles that correspond to graphs and graph morphisms. Then we define a tile rewrite system based on such arrows.

The following definition is a slight restriction of the definition in [17] to consider only undirected hypergraphs.

**Definition 6 (One-sorted (hyper-)signatures).** *A **hyper-signature** $\Sigma = (S, OP)$ consists of a singleton $S = \{s\}$ and a family $OP = \{op_h\}_{h \in \mathbb{N}}$ of sets of* operators *with $h$ arguments and no result (no target sort), where $h$ is a natural number. Given an operation $op_h \in OP$, $h$ is the arity of $op_h$, and the domain of $op_h$ is denoted by $s^h$. $\Sigma_n$ denotes the set of operators of $OP$ with arity $n$.*

Now we define an extension of a signature $\Sigma$ that will add as sorts all operation names in $\Sigma$ and as target of each operation the corresponding sort.

**Definition 7 (Extended (hyper-)signatures).** *Given a signature $\Sigma = (S, OP)$, its **extension** is a signature $\Sigma^E = (S_E, OP_E)$, where $S_E = \{\underline{s}\} \cup \{\underline{op} | op : n \to 0 \in OP\}$ and $OP_E = \{op_h^E : \underline{s^h} \to \underline{op_h} | op_h : s^h \to 0 \in OP\}$.*

### 3.1 Monoidal Theories

We will define algebraic theories to describe graphs and graph morphisms. In particular, the gs-monoidal theory defined here is an extension of the one defined in [4] because we consider different arrows for sorts of an extended hyper signature: for the only sort coming from the underlying hyper signature, we use the usual definition of gs-monoidal theories, but for the other ones, we do not allow the duplicator and discharger operators. In addition, we define a new theory: the partial graph morphism (pgm) theory, allowing discharger and new operators for sorts that are not in the underlying signature only (the rest is the same as in the gs-monoidal theory). The reason for this will become clear in the next section, where we will use gs-monoidal theories to model (hyper)graphs and pgm-monoidal theories to model special (hyper)graph morphisms.

**Definition 8 (Connection diagrams).** *A* connection diagram *$G$ is a 4-tuple $\langle O_G, A_G, \delta_0, \delta_1 \rangle$: $O_G$, $A_G$ are sets whose elements are called respectively* objects *and* arrows*, and $\delta_0, \delta_1 : A_G \to O_G$ are functions, called respectively* source *and* target*. A connection diagram $G$ is* reflexive *if there exists an* identity *function $id_- : O_G \to A_G$ such that $\delta_0(id_a) = \delta_1(id_a) = a$ for all $a \in O_G$; it is* with pairing *if its class $O_G$ of objects forms a monoid; it is* monoidal *if it is reflexive with pairing and also its class of arrows forms a monoid, such that $id$, $\delta_0$ and $\delta_1$ respect the neutral element and the monoidal operation.*

In the following we will use the free commutative monoid construction over a set $S$, denoted by $(S^\otimes, \otimes, \underline{0})$. The elements of $S^\otimes$ can be seen as lists of elements of $S$, and will be referred to using underlined variables. If $S$ is a singleton set, we will denote elements of $S^\otimes$ by underlined natural numbers (indicating the number of times the only element of $S$ appears in the list, for example $\underline{s} \otimes \underline{s} \otimes \underline{s}$ will be denoted by $\underline{3}$).

**Definition 9 (Graph theories).** *Given a (hyper)signature $\Sigma = (S, OP)$ and its extension $\Sigma^E = (S_E, OP_E)$, the associated* graph theory *$G(\Sigma)$ is the monoidal connection diagram with objects the elements of the free commutative monoid over $S_E$ $((S_E)^{\otimes}, \otimes, \underline{0})$ and arrows generated by the inference rules* generators, sum *and* identities *shown in Table 1 satisfying the* monoidality *axiom $id_{\underline{x} \otimes \underline{y}} = id_{\underline{x}} \otimes id_{\underline{y}}$ for all $\underline{x}, \underline{y} \in S_E^{\otimes}$.*

**Table 1.** Inference Rules

$$(generators)\ \frac{f \in \Sigma_n}{f : \underline{n} \to \underline{f} \in G(\Sigma)} \qquad (sum)\ \frac{s : \underline{x} \to \underline{y}, t : \underline{x'} \to \underline{y'}}{s \otimes t : \underline{x} \otimes \underline{x'} \to \underline{y} \otimes \underline{y'} \in G(\Sigma)}$$

$$(identities)\ \frac{\underline{x} \in S_E^{\otimes}}{id_{\underline{x}} : \underline{x} \to \underline{x} \in G(\Sigma)}$$

---

$$(composition)\ \frac{s : \underline{x} \to \underline{y}, t : \underline{y} \to \underline{z}}{s; t : \underline{x} \to \underline{z} \in \mathbf{M}(\Sigma)}$$

---

$$(duplicators)\ \frac{\underline{n} \in S^{\otimes}}{\nabla_{\underline{n}} : \underline{n} \to \underline{n} \otimes \underline{n} \in \mathbf{GS}(\Sigma)} \qquad (dischargers)\ \frac{\underline{n} \in S^{\otimes}}{!_{\underline{n}} : \underline{n} \to \underline{0} \in \mathbf{GS}(\Sigma)}$$

$$(permutations)\ \frac{\underline{x}, \underline{y} \in S_E^{\otimes}}{\rho_{\underline{x},\underline{y}} : \underline{x} \otimes \underline{y} \to \underline{y} \otimes \underline{x} \in \mathbf{GS}(\Sigma)}$$

**Definition 10 (Monoidal theories).** *Given a (hyper)signature $\Sigma = (S, OP)$ and its extension $\Sigma^E = (S_E, OP_E)$, the associated* monoidal theory *$\mathbf{M}(\Sigma)$ is the monoidal connection diagram with objects the elements of the free commutative monoid over $S_E$ $((S_E)^{\otimes}, \otimes, \underline{0})$ and arrows generated by the following inference rules:* generators, sum *and* identities *(analogous to the rules in Def. 9), and the rule* composition *in Table 1. Moreover, the composition operator ; is associative, and the monoid of arrows satisfies the* functoriality *axiom*

$$(s \otimes t); (s' \otimes t') = (s; s') \otimes (t; t')$$

*(whenever both sides are defined) and the* identity *axiom $id_{\underline{x}}; s = s = s; id_{\underline{y}}$ for all $s : \underline{x} \to \underline{y}$.*

**Definition 11 (gs-monoidal theories).** *Given a (hyper)signature $\Sigma = (S, OP)$ and its extension $\Sigma^E = (S_E, OP_E)$, the associated* ***gs-monoidal theory*** *$\mathbf{GS}(\Sigma)$ is the monoidal connection diagram with objects the elements of the free commutative monoid over $S_E$ $((S_E)^{\otimes}, \otimes, \underline{0})$ and arrows generated by the following inference rules:* generators, sum, identities *and* composition *(analogous to the rules in Def. 10), and the rules* duplicators, dischargers *and* permutations *in Table 1. Moreover, the composition operator ; is associative, and the monoid of arrows satisfies the functoriality axiom (see Def. 10); the identity axiom (see Def. 10); the monoidality axioms: for all $\underline{n}, \underline{m} \in S^{\otimes}$ and $\underline{x}, \underline{y}, \underline{z} \in S_E^{\otimes}$*

$$\rho_{\underline{x} \otimes \underline{y}, \underline{z}} = (id_{\underline{x}} \otimes \rho_{\underline{y},\underline{z}}); (\rho_{\underline{x},\underline{z}} \otimes id_{\underline{y}}) \qquad !_{\underline{0}} = \nabla_{\underline{0}} = \rho_{\underline{0},\underline{0}} = id_{\underline{0}} \qquad \rho_{\underline{0},\underline{x}} = \rho_{\underline{x},\underline{0}} = id_{\underline{x}}$$

$$!_{\underline{x} \otimes \underline{y}} = !_{\underline{x}} \otimes !_{\underline{y}} \qquad \nabla_{\underline{n} \otimes \underline{m}} = (\nabla_{\underline{n}} \otimes \nabla_{\underline{m}}); (id_{\underline{n}} \otimes \rho_{\underline{n},\underline{m}} \otimes id_{\underline{m}})$$

*the* coherence *axioms: for all* $\underline{n} \in S^{\otimes}$ *and* $\underline{x}, \underline{y} \in S_E^{\otimes}$

$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes \nabla_{\underline{n}}) = \nabla_{\underline{n}}; (\nabla_{\underline{n}} \otimes id_{\underline{n}}) \qquad \nabla_{\underline{n}}; \rho_{\underline{n},\underline{n}} = \nabla_{\underline{n}}$$
$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes !_{\underline{n}}) = id_{\underline{n}} \qquad \rho_{\underline{x},\underline{y}}; \rho_{\underline{x},\underline{y}} = id_{\underline{x}} \otimes id_{\underline{y}}$$

*and the* naturality *axiom: for all* $s : \underline{x} \to \underline{y}, t : \underline{z} \to \underline{w}$

$$(s \otimes t); \rho_{\underline{y},\underline{w}} = \rho_{\underline{x},\underline{z}}; (t \otimes s)$$

*Arrows of gs-monoidal theories that are constructed without the* generators *axiom are called* ***basic gs-monoidal arrows****. The theory obtained only with basic arrows is denoted by* $\mathbf{bGS}(\Sigma)$.

In [4] it was shown that term graphs (a particular kind of hierachical hypergraphs) could be suitably represented as arrows of a gs-monoidal theory. Here we are rather interested in non-hierachical graphs, and the restriction is achieved by selecting signatures where operators have output types which are disjoint from input types, i.e. operators cannot be nested. Objects represent lists of variables (nodes) and an arrow $\underline{n} \to \underline{m}$ represents the application of a (multi)set of operators (arcs) to the variables $\underline{n}$ generating the results (roots) $\underline{m}$. For example, assuming that the sort of nodes of $L$ is $\{s\}$, the graph $L$ in Figure 1 represents the union of term-graphs $A(\bullet_2, \bullet_1)$ and $B(\bullet_1)$ sharing the variable $\bullet_1$. This term-graph can be modeled as the following gs-monoidal arrow: $\rho_{\underline{s},\underline{s}}; (id_{\underline{s}} \otimes \nabla_1); (A \otimes B) : \underline{s} \otimes \underline{s} \to \underline{A} \otimes \underline{B}$. Graphically, this arrow is represented in Figure 2 (1 and 2 are the first and second occurrences of sort $s$ in $\underline{s} \otimes \underline{s}$).

*Notation:* In the pictures, the operator $\otimes$ and the underlines in the domain and codomain of the arrows will be omitted. Moreover, we will draw indexed bullets ($\bullet$) in the graphical representation to describe the instances of sorts.

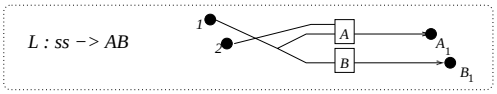


**Fig. 2.** Arrow of a gs-monoidal theory

The theory defined below will be used to model a special kind of graph morphism: morphisms that are total on nodes (and may identify nodes), and are partial on arcs (and may not identify arcs). Identification of nodes is achieved by the $\nabla$ operator, as in gs-monoidal theories. The possibility of deleting arcs is achieved by the $\mathsf{i}$ operator, yielding partial functions (only for arcs).

**Definition 12 (pgm-monoidal theories).** *Given a (hyper)signature* $\Sigma = (S, OP)$ *and its extension* $\Sigma^E = (S_E, OP_E)$*, the associated* ***pgm-monoidal theory*** $\mathbf{PGM}(\Sigma)$ *is the monoidal connection diagram with objects the elements of the free commutative monoid over* $S_E$ $((S_E)^{\otimes}, \otimes, \underline{0})$ *and arrows generated by the following inference rules:* sum, identities, composition *and* permutations *(analogous to the rules in Def. 11), and*

$$\text{(new)}\ \frac{\underline{x} \in (S_E - S)^{\otimes}}{\mathsf{i}_{\underline{x}} : \underline{0} \to \underline{x} \in \mathbf{PGM}(\Sigma)}$$

$$\text{(duplicators)}\ \frac{\underline{n} \in S^{\otimes}}{\nabla_{\underline{n}} : \underline{n} \to \underline{n} \otimes \underline{n} \in \mathbf{PGM}(\Sigma)} \qquad \text{(dischargers)}\ \frac{\underline{x} \in S_E^{\otimes}}{!_{\underline{x}} : \underline{x} \to \underline{0} \in \mathbf{PGM}(\Sigma)}$$

*The composition operator* ; *is associative and the monoid of arrows satisfies the* functoriality *axiom (Def. 10); the* identity *axiom (Def. 10); the* monoidality *axioms (all of Def. 11 plus* $!_{\underline{0}} = \mathsf{i}_{\underline{0}} = \rho_{\underline{0},\underline{0}} = id_{\underline{0}}$ *and* $\mathsf{i}_{\underline{x}\otimes\underline{y}} = \mathsf{i}_{\underline{x}} \otimes \mathsf{i}_{\underline{y}}$*, for all* $\underline{x}$, $\underline{y}$, $\underline{z} \in S_E^{\otimes}$*); the* coherence *axiom (Def. 11); and the* naturality *axiom (Def. 11). The theory obtained using the same objects and arrows generated by the rules above, except the* composition *is called* ***short pgm-graph theory****, denoted by* $\mathbf{sPGM}(\Sigma)$*. The theory obtained using all rules except* new *and with* discharger *only for* $\underline{x} \in S^{\otimes}$ *is called* ***basic pgm-graph theory****, denoted by* $\mathbf{bPGM}(\Sigma)$*.*

Note that in pgm-monoidal theories we did not use the *generators* axiom. This has the effect that arrows of these theories do not correspond to graphs as gs-monoidal theories (because arcs are not allowed), they rather describe relationships between the objects (that are lists of nodes and arc labels). Moreover, an arrow of $\mathbf{bPGM}(\Sigma)$ is also an arrow of $\mathbf{bGS}(\Sigma)$.

In the following definition, we use the fact that basic gs-monoidal arrows of one sorted signatures correspond to total functions in the inverse direction, that is, each gs-monoidal arrow $\underline{n} \to \underline{m}$ corresponds to a total function $\underline{m} \to \underline{n}$ (see [4] for the proof). As basic pgm-monoidal arrows are also basic gs-monoidal arrows, they also correspond to total functions. This will be used to construct pullback squares of such arrows based on pushouts of functions. These pushouts will model the node-component morphisms of rule applications.

**Definition 13 (Derivation Pair).** *Given one-sorted signatures* $\Sigma_h$ *and* $\Sigma_v$*, and arrows* $s : \underline{n} \to \underline{m} \in \mathbf{bGS}(\Sigma_h)$ *and* $t : \underline{q} \to \underline{m} \in \mathbf{bPGM}(\Sigma_v)$*. The* ***derivation pair*** *of* $s$ *and* $t$*, denoted by* $deriPair(s,t)$*, is a pair of a basic gs-monoidal and a basic pgm-monoidal arrows* $(s' : \underline{p} \to \underline{q}, t' : \underline{p} \to \underline{n})$ *such that the inverse square is a pushout in the category of sets and total functions.*

### 3.2 Tile Rewriting Systems

**Definition 14 (Tile sequent, Tile rewrite system).** *Let* $\Sigma_h$ *and* $\Sigma_v$ *be two (one sorted) signatures, called the* horizontal *and the* vertical *signature respectively. A* $\Sigma_h$-$\Sigma_v$ tile sequent *is a quadruple* $l \xrightarrow[b]{a} r$*, where* $l : \underline{x} \to \underline{y}$ *and* $r : \underline{p} \to \underline{q}$ *are arrows of* $\mathbf{GS}(\Sigma_h)$ *while* $a : \underline{x} \to \underline{p}$ *and* $b : \underline{y} \to \underline{q}$ *are arrows of* $\mathbf{PGM}(\Sigma_v)$*. Arrows* $l$*,* $r$*,* $a$ *and* $b$ *are called respectively the* initial configuration*, the* final configuration*, the* trigger *and the* effect *of the tile. Trigger and effect are called* observations*. Underlined strings* $\underline{x}$*,* $\underline{y}$*,* $\underline{p}$ *and* $\underline{q}$ *are called respectively the* initial input interface*, the* initial output interface*, the* final input interface *and the* final output interface*.*

*A* short *tile sequent is a tile* $l \xrightarrow[b]{a} r$ *where observations* $a$ *and* $b$ *are arrows of the short pgm-graph theory* $\mathbf{sPGM}(\Sigma_v)$ *(i.e. no sequential composition is allowed to build them).*

*A* tile rewrite system (TRS) $\mathcal{R}$ *is a triple* $\langle \Sigma_h, \Sigma_v, R \rangle$*, where* $R$ *is a set of* $\Sigma_h$-$\Sigma_v$ *short tile sequents called* rewrite rules*.*

A TRS $\mathcal{R}$ can be considered as a logical theory, and new sequents can be derived from it via certain inference rules.

**Definition 15 (pgm-monoidal tile logic).** *Let $\mathcal{R} = \langle \Sigma_h, \Sigma_v, R \rangle$ be a* TRS. *Then we say that $\mathcal{R}$* entails *the class $\mathbf{R}$ of the* tile sequents $s \overset{a}{\underset{b}{\to}} t$ *obtained by finitely many applications of the inference rules depicted in Table 2.*

**Table 2.** pgm-monoidal Tile Logic

*Basic rules:*

$$(generators)\ \frac{s \overset{a}{\underset{b}{\to}} t \in \mathcal{R}}{s \overset{a}{\underset{b}{\to}} t \in \mathbf{R}} \quad (h\text{-}refl)\ \frac{s : \underline{x} \to \underline{y} \in \mathbf{GS}(\Sigma_h)}{id_s = s \overset{id_{\underline{x}}}{\underset{id_{\underline{y}}}{\to}} s \in \mathbf{R}} \quad (v\text{-}refl)\ \frac{a : \underline{x} \to \underline{y} \in \mathbf{PGM}(\Sigma_v)}{id_a = id_{\underline{x}} \overset{a}{\underset{a}{\to}} id_{\underline{y}} \in \mathbf{R}};$$

*Composition rules:*

$$(p\text{-}comp)\ \frac{\alpha = s \overset{a}{\underset{b}{\to}} t, \alpha' = s' \overset{a'}{\underset{b'}{\to}} t' \in \mathbf{R}}{\alpha \otimes \alpha' = s \otimes s' \overset{a \otimes a'}{\underset{b \otimes b'}{\to}} t \otimes t' \in \mathbf{R}}$$

$$(h\text{-}comp)\ \frac{\alpha = s \overset{a}{\underset{c}{\to}} t, \alpha' = s' \overset{c}{\underset{b}{\to}} t' \in \mathbf{R}}{\alpha * \alpha' = s; s' \overset{a}{\underset{b}{\to}} t; t' \in \mathbf{R}} \quad (v\text{-}comp)\ \frac{\alpha = s \overset{a}{\underset{b}{\to}} u, \alpha' = u \overset{a'}{\underset{b'}{\to}} t \in \mathbf{R}}{\alpha \cdot \alpha' = s \overset{a;a'}{\underset{b;b'}{\to}} t \in \mathbf{R}};$$

*Auxiliary rules:*

$$(perm)\ \frac{a : \underline{x} \to \underline{y}, b : \underline{x'} \to \underline{y'} \in \mathbf{PGM}(\Sigma_v)}{\rho_{a,b} = \rho_{\underline{x},\underline{x'}} \overset{a \otimes b}{\underset{b \otimes a}{\to}} \rho_{\underline{y},\underline{y'}} \in \mathbf{R}}$$

$$(PBnodes)\ \frac{s : \underline{n} \to \underline{m} \in \mathbf{bGS}(\Sigma_h), a : \underline{q} \to \underline{m} \in \mathbf{bPGM}(\Sigma_v)}{\underline{s} \overset{a'}{\underset{a}{\to}} \underline{s'} \in \mathbf{R}} (s', a') = deriPair(s, a)$$

## 4 Linear Ordered Graph Grammars

Now we will define a restricted type of graph grammar: the nodes and arcs of each graph are ordered, the rules do not delete nodes and do not collapse arcs, and the matches do not collapse arcs. With this kind of restriction, the derivation steps can be obtained componentwise as a pushout of nodes and pushout of arcs (because no nodes are deleted).

The motivation for the definition of this kind of graph grammar is mainly based on the applications we have in mind: mobile code and object-based systems. The idea is to model static components (places, communication channels, objects) as nodes and resources (processes, agents, ambients, messages) as arcs. In this view, it is natural to require that we have a different way to handle these two kinds of entities, and that is why we have a distinct treatment of nodes and arcs in a linear ordered graph grammar. The fact that nodes and arcs are ordered was imposed to achieve a suitable true concurrency semantics. In the unordered case, the description of concurrency is analogous to the collective view of tokens in a Petri net, that leads to a different semantical model as the individual token philosophy. The latter is the standard way of considering the semantics of graph rewriting. A good comparison of the approaches for the case of Petri nets can be found in [3]. To illustrate this situation, consider the grammar depicted in Figure 3. There is only one rule, that deletes and creates arcs of type $B$. The start graph of the grammar has two $B$-arcs. Some possible behaviors of this grammar are processes $P1$ and $P2$, that replace, using rule $r$, the $B_1$ and $B_2$-arcs of $IG$ respectively by other $B$-arcs. If we consider that arcs are not ordered, these processes are isomorphic. Let process $P3$ be the concatenation of $P1$ and $P2$. This

process has two applications of rule $r$, but we can not distinguish whether these applications are dependent or independent, that is, the process in which the second application of $r$ depends on the first is equivalent to the one in which they are independent. This contradicts the basic idea of a true concurrency semantics, in which dependencies between actions are essential.

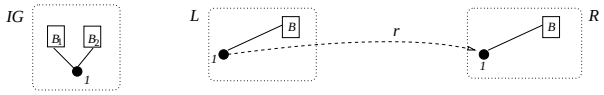


**Fig. 3.** Start Graph $IG$ and Rule $r$

**Definition 16 (Linear Ordered Graph Grammars).** *A **linear ordered (hyper)graph** over a type graph $TG = (\{s\}, A_{TG}, source^{TG})$ is a typed hypergraph $HT^{TG}$ with $HG = (N, A_{HG}, source^{HG})$ where $N = \{0..n\}$, with $n \in \mathbb{N}$, and $A_{HG} = \bigcup_{a \in A_{TG}} A^a_{HG}$, with $A^a_{HG} = \{a_i | i \in \{0..m\}\}$, where $m$ is the cardinality of $\{ha \in A_{HG} | type^{HG}(ha) = a\}$. A morphism between linear ordered hypergraphs is simply any morphism typed hypergraph morphism.*

*A **linear ordered graph grammar** is a graph grammar $GG = (TG, IG, Rules)$ where $TG$ has only one type of nodes, $IG$ and all graphs in the rules of $Rules$ are linear ordered graphs and for each rule $r = (r_N, r_A) : L \to R$ of $Rules$, $r_A$ is injective and $r_N$ is total. A **linear ordered match** $m = (m_N, m_A)$ is a total graph morphism where the $m_A$ component is injective.*

The semantics of a linear ordered graph grammar is defined as for a usual graph grammar, taking into account that only linear ordered matches are allowed, that is, a match is resource conscious on arcs, but may identify nodes.

In the following sections, we will show how to describe the syntax and semantics of such a graph grammar using tiles and tile rewriting systems.

### 4.1 Syntax: Graphs and Rules

Consider the linear ordered derivation step shown in Figure 1. The graphs used in this derivation are hypergraphs having only one type of node and three types of (hyper)arcs, having the following arities (consider the sort of nodes of $TG$ to be $s$): $A : s \times s, B : s, C : s \times s$. Our idea of representing graphs and rules as tiles is as follows:

**Graphs:** Graphs will be modeled as the horizontal components of the tiles. Let $TG = (\{s\}, A, source^{TG})$ be a type graph. Based on this graph we can build a siganture $\Sigma_h = (\{s\}, A_\Sigma)$, where $A_\Sigma$ contains all arcs of $A$ as operations (the information about the arity is given by the $source^{TG}$ function). A graph will be then an arrow $\underline{x} \to \underline{y}$ of the corresponding gs-monoidal theory, where $\underline{x}, \underline{y} \in (\{s\} \uplus A)^\otimes$. This means that the mapping from $\underline{x}$ to $\underline{y}$ is constructed by using, besides the arcs of the graph, the operations of identity and permutation for all sorts, and for elements of sort $s$ (nodes), we additionally allow duplication and discharging, to allow that the same node may be used as source of many hyperarcs, and that it may not be used by any hyperarc. For the hyperarcs of the graph we include a target function that assigns to each arc an occurrence of its type (indexed by a natural number). For example,

the graph $L$ can be described by the gs-monoidal arrow illustrated in Figure 2. Usual (closed) graphs correspond to the special case of $G : \underline{x} \to \underline{y}$ when we have $\underline{x} \in \{s\}^{\otimes}$ and $\underline{y} \in A^{\otimes}$. Other graphs are called *open graphs*, and will be used as auxiliary components to allow the modeling of the direct derivation.

**Rules:** A rule $r : L \to R$ is a (partial) graph morphism $r = (r^N, r^A)$. Such a rule can be represented as a tile having as horizontal arrows the graphs $L$ and $R$, and as vertical arrows mappings that allow to glue nodes, delete arcs, preserve and create arcs and nodes. These vertical arrows are arrows of the pgm-monoidal theory for the signature $\Sigma_v = \Sigma_h$. Rule $r$ of Figure 1 corresponds to the tile of Figure 4 (vertical mapping is shown as dashed arrows, creation - new - is denoted by the i, and deletion - bang - is denoted by !). Note that at the west side of the tile $r$ we have modeled the component $r^N$ of the rule, whereas in the east side we model the component $r^A$.

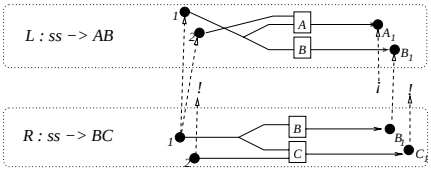


**Fig. 4.** Modeling a rule as a tile

**Definition 17 (Signature of a Type Graph).** *Given a type graph* $TG = (N, A, source^{TG})$*, the corresponding hypergraph signature is* $\Sigma_{TG} = (N, A_{TG})$*, where* $A_{TG} = \{a : source^{TG}(a) \to \underline{0} | a \in A\}$.

The characterization below describes which arrows of a gs-monoidal theory correspond to graphs. Other arrows will be called *open graphs*. The proof was omitted due to space limitations (similar proofs can be found in [4] and [17]).

**Proposition 1 (Characterization of graphs).** *The graphs typed over* $TG = (N, A, source^{TG})$ *with* $N$ *being a singleton are the arrows of kind* $\underline{n} \to \underline{a}$ *of a gs-monoidal theory with signature* $\Sigma_{TG}$*, where* $\underline{n} \in N^{\otimes}$ *and* $\underline{a} \in (N^E - N)^{\otimes}$*, with* $\Sigma_{TG}^E = (N^E, A^E)$.

The characterization below describes a kind of graph morphism that may identify nodes but not arcs, and may be partial on arcs, but not on nodes. The rules of a linear ordered graph grammar are exactly this kind of morphisms.

**Proposition 2 (Characterization of graph morphisms).** *Graph morphisms typed over* $TG$ *are tiles having as horizontal and vertical signature* $\Sigma_{TG}$ *where the north and south sides are the arrows corresponding to (closed) graphs (the left- and right-hand sides of the rule, respectively), and the vertical arrows are arrows of* $\mathbf{PGM}(\Sigma_{TG})$.

*Proof.* An SPO-rule consists of a left-hand side $L$, a right-hand side $R$ and a partial graph morphism $r = (r_N, r_A) : L \to R$. Consider the gs-monoidal arrows $s_L : \underline{n} \to \underline{a}$ and $s_R : \underline{n'} \to \underline{a'}$ corresponding to graphs $L$ and $R$, respectively. As $r_N$ is a total function, there is a corresponding gs-monoidal arrow $b_N : \underline{n'} \to \underline{n}$ mapping the nodes of $R$ into the nodes of $L$. Arrow $b_N \in \mathbf{PGM}(\Sigma_{TG})$ because

$r_N$ is just a mapping of nodes, and this can be modeled just by using identities, duplicators, discharger and permutation operators for the sort of nodes, and the corresponding rules may be used to build $\mathbf{PGM}(\Sigma_{TG})$. The function that maps arcs of $L$ into arcs of $R$ does not allow the identification of arcs, but allows to delete and create arcs. Therefore, we need identities, dischargers, new and permutation operators for the sorts of arcs, and corresponding rules are allowed to build $\mathbf{PGM}(\Sigma_{TG})$. The other direction can be proven analogously.

### 4.2 Semantics

Now we will show how a direct derivation of a grammar $GG$ can be modeled by a suitable composition of tiles of the pgm-monoidal tile logic $\mathbf{R}$ obtained by the TRS $\langle \Sigma_h, \Sigma_v, R \rangle$, where $\Sigma_h = \Sigma_v = \Sigma_{TG}$ (as discussed above), and $R$ is the set of tiles representing the rules of $GG$. A derivation $G \overset{r,m}{\Longrightarrow} H$ using rule $r : L \to R$ at match $m : L \to G$ can be obtained as a composition of tiles that will give raise to a tile $r' : G \to H$. This can be done in 4 steps:

1. **Context Graph**: Construct the (context) graph $G'$, that contains all nodes of $G$ and all arcs that are in $G$ and are not in the image of the match $m$ (that is, $G'$ is $G$ after removing the deleted and preserved arcs). Note that $G'$ must be a graph because we do not allow rules that delete nodes (and therefore $G'$ can not contain dangling edges). For the example derivation in Figure 1 we would have the graphs corresponding to $G$ and $G'$ in Figure 5 (note that the arc $A_1$ of $G'$ corresponds to the arc $A_2$ of $G$ – that name in $G'$ must have the index 1 because it is the first occurrence of sort $A$ in $AC$).

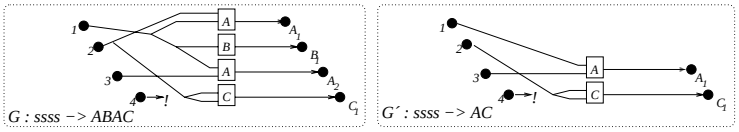


**Fig. 5.** Graph $G$ and context graph $G'$

2. **Tile 1**: Construct the tile $r \otimes id_{G'}$, that is the parallel composition of the tile corresponding to rule $r$ and the identity tile of graph $G'$. Note that, as $L$ and $G$ are graphs, the west side of this tile has only nodes, whereas the east side has only arcs. This tile belongs to $\mathbf{R}$ because rule $r \in \mathbf{R}$ (rule *generators*), $id'_G \in \mathbf{R}$ (rule *h-refl*) and thus the composed tile also belongs to $\mathbf{R}$ (rule *p-comp*). In the example we will get the tile (a) in Figure 6.
3. **Tile 2**: Construct the tile corresponding to the match and derivation on the node component: the north side of this tile corresponds to $m^N$ (the component of the match morphism that maps nodes), the east side is the west side of the tile obtained in step 2, and the remaining sides will be mappings of nodes such the the resulting square commutes and has no nodes that are not in the north or east sides already. Considering that the algebraic structures used to build these mappings are functions in the inverse direction, this square will be a pushout in the category of sets and partial functions (in this case, all functions are actually total because deleting of nodes is not allowed in our setting). This tile belongs to $\mathbf{R}$ due to rule *PBnodes*, that says

that all such pushouts (considered as pullbacks in the opposite direction) are auxiliary tiles of **R**. In our example, the tile obtained by this construction is shown in Figure 6 (b).

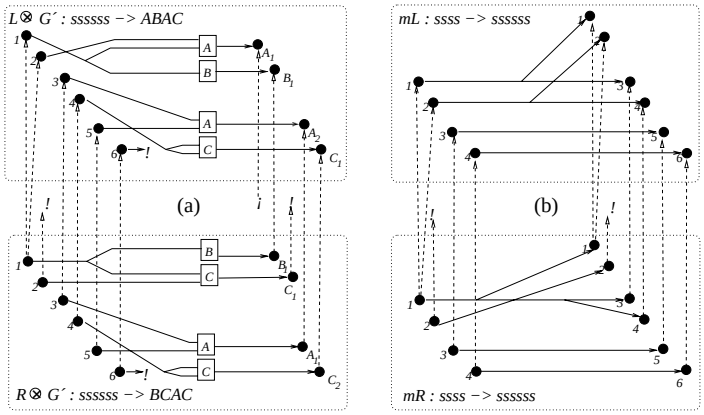


**Fig. 6.** (a) Tile 1 (b) Tile 2

4. **Resulting Tile**: The result of the application of rule $r$ at match $m$ is then given by the sequential composition of the tiles obtained in the last 2 steps: $Tile\ 1 * Tile\ 2$ (obtained by the application of rule *h-comp* of **R**). In the example, this is shown in Figure 7.

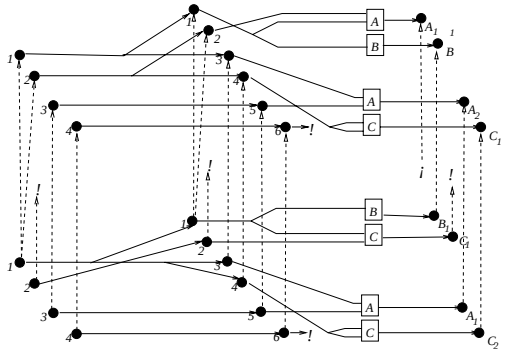

**Fig. 7.** Resulting tile

Note that we did not need to construct an auxiliary tile that corresponds to the pushout of arcs, as we did for the nodes (in tile 2). This is due to the fact that we do not allow to identify arcs neither in the rule nor in the match, and therefore the east-south component of tile 1 gives us already the arcs of the resulting graph of the derivation. In the classical SPO-setting, the corresponding pushout of arcs would have exactly the same effect: it would put all arcs that are in the right-hand side together all arcs that are in $G$ and not in the image of the left-hand side of the rule (because the rule and the match are injective on arcs and the match is total). In case we would allow identification of arcs we would need a step for arcs corresponding to step 2.

**Definition 18 (Algebraic Semantics of a Linear Ordered Graph Grammar).** *Given a graph grammar $GG = (TG, IG, Rules)$, its* ***algebraic semantics****, denoted by $Alg(GG)$, is given by the tile rewrite system $\mathcal{R} = \langle \Sigma_{TG}, \Sigma_{TG}, R \rangle$, $R$ is the set of tiles corresponding to the rules of $GG$.*

**Proposition 3.** *Let $\sigma = G_0 \overset{r_1,m_1}{\Longrightarrow} G1 \Rightarrow \cdots \overset{r_n,r_m}{\Longrightarrow} G_n$ be a sequential derivation of a graph grammar $GG$. Then there is a tile $s \overset{a}{\underset{b}{\rightarrow}} t$ in $Alg(GG)$ such that $s$ and $t$ are the graphs corresponding to $G_0$ and $G_n$, respectively, and $(a, b)$ corresponds to the derived rule of the sequential derivation $\sigma$.*

*Proof sketch.* As discussed above, all derivation steps of a grammar can be constructed using the tile corresponding to the rule, identity tiles and pullback tiles of nodes, composed using parallel and sequential composition. According to the definition of tile rewrite system, these tiles and operations can be used to build composed tiles, and therefore a tile corresponding to the derived rule must be in this tile rewrite system. Moreover, vertical composition of tiles is allowed, yielding the composition of co-rules. Note that the vertical composition of tiles is done by composing the west and east sides, and these sides correspond to the mapping of nodes and arcs of the involved graphs, respectively. The derived rule obtained by the composition $rn' \circ \cdots \circ r1'$ is also constructed componentwise (composition of partial graph morphisms).

**Proposition 4.** *Given a graph grammar and a tile rewrite system $Alg(GG)$, the horizontal tiles of $Alg(GG)$ that use one rule of $GG$ (one generator tile) and have graphs as north and south sides correspond to derivation rules of $GG$.*

*Remark*: Horizontal tiles are the ones that are built without using the rule for vertical composition. If we allow more than one rule, the resulting tile would correspond to a parallel application of rules.

*Proof sketch.* Assume that a tile $s \overset{a}{\underset{b}{\rightarrow}} t$ uses one rule of $GG$ and has graphs as north and south sides. If this tile is not obtained by composition of other tiles, it must be a generator, that is, a rule of $GG$. Generators correspond to rules of $GG$ and thus such tiles correspond to derivations of kind $L \overset{r,id_L}{\Longrightarrow} R$, with $r : L \rightarrow R$. All tiles using tile $Tr$ corresponding to rule $r$ can obtained by composition can be rearranged (using the axioms) to the form $T1; (T2 \otimes Tr \otimes T3); T4$, where $T1$ has only nodes, $T2$ and $T3$ have graphs as north and south sides, and $T4$ has only arcs (elements of sorts $Ai$). Thus, $T1$ must be a PBnode tile because it must be the composition of identities and PBnode tiles (due to compositionality of colimits, the resulting tile must be a PBnode). $T2 \otimes T3$ can be seen as the context graph to which the rule is applied to. As $T2$ and $T3$ must be closed graphs, they can only be built by parallel composition of other closed graphs, and therefore must be compositions of identities (they can not be rules because $Tr$ is the only rule in this tile, and can not be PBnodes or permutations of nodes/arcs because all these tiles have been rearranged to the component $T1/T4$). If $T4$ is not empty, it is just a permutation of arcs, depending on which permutation we take, we get all different isomorphic results of a direct derivation (on the arc component, on the node component this correspond to using permutation tiles to build $T1$).

## 5 Conclusion and Future Work

In this paper we defined a special kind of grammars, called linear ordered graph grammars, that can be used to describe distributed systems with mobility and object-based systems. We showed how to model such grammars and their semantics in terms of tiles. In [11] graph grammars in the DPO approach have been modeled using tiles. The idea was that horizontal arrows correspond to graph morphisms, vertical arrows are pairs of an identity (on the gluing graph) and the name of a production and tiles correspond to DPO rules. In this way, the observations (vertical components of a tile) are not put into foreground (they are just names of rules). In our approach, we have modeled a graph morphism as vertical component of a tile. We could easily extend our approach to consider different kinds of arcs (observable and non-observable) corresponding to different horizontal and vertical signatures. This way we could model, for example, transactions, analogously to what have been done for Petri nets in [1]. The main aspects of our approach are:

• Types in graphs correspond to signatures in tiles: typed graphs correspond to gs-monoidal arrows of a restricted type on a signature. Graph isomorphism is directly represented by gs-monoidal axioms. Moreover, the set of all arrows has a natural meaning in terms of open graphs (due to space limitations, this topic was not further detailed here, see [17]).

• Graph morphisms of the special kind we need are obtained by choosing the vertical structure of tiles adequately. It is immediately apparent the correspondence between the vertical "wire structure" (again a variation of symmetric monoidal categories) and the properties of morphisms.

• A SPO can be represented by a horizontal tile construction. Again, the pushout property becomes concrete in terms of certain tiles which can be constructed starting from a finite number of auxiliary tiles [2].

• The effect of a derivation can be represented by a (flat) tile. We expect that canonical derivations are exactly represented by tiles equipped with their proof terms (in the Curry-Howard style). Note that we did not restrict to injective rules (as it is done in almost all true concurrency models for graph grammars).

• The observable effect of a derivation (which is not usually emphasized in the SPO/DPO literature) is very important in the interactive interpretation of our models of computations [2]. It represents the creation/fusions of channels and creation/termination of processes which took place in the computation. Typically, a useful way of connecting systems is via composition of their states, that can be done through shared channels. Therefore it is important to have information about these channels in interfaces (that is, make them observable) to obtain a compositional semantics.

• The model would be more satisfactory introducing also restriction/hiding of channels and processes, which would be straightforward (as shown by other papers [9]) in the tile model. This would correspond to new and interesting classes of graphs, making even more explicit the advantage of establishing a connection between graph transformations and tiles.

## References


1. R. Bruni and U. Montanari, *Zero-Safe Nets: Comparing the Collective and Individual Token Approaches*, Information and Computation, Vol. 156, 2000, pp. 46–89.
2. R. Bruni, U. Montanari and F. Rossi, *An Interactive Semantics of Logic Programming*, TLP 6 (1): 647-690, Nov. 2001.
3. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone, *Functorial models for Petri nets*, Information and Computation, Vol. 170, 2001, pp. 207–236.
4. A. Corradini and F. Gadducci, *An Algebraic Presentation of Term Graphs via Gs-Monoidal Categories*, Applied Categorical Structures, Vol. 7, 1999, pp. 299–331.
5. A. Corradini, U. Montanari and F. Rossi, *Graph processes*, Fundamentae Informatica, Vol. 26, no. 3-4, 1996, pp. 241–265.
6. F. Dotti and L. Ribeiro, *Specification of mobile code systems using graph grammars*, Formal Methods for Open Object-based Systems IV, Kluwer Academic Publishers, 2000, pp.45–64.
7. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, World Scientific, 1999.
8. H. Ehrig, R. Heckel, M. Korff, M. Lwe, L. Ribeiro, A. Wagner and A. Corradini, *Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach*, in [18], pp. 247–312.
9. G. Ferrari and U. Montanari, *Tile Formats for Located and Mobile Systems*, Information and Computation, Vol. 156, no. 1/2, 2000, pp. 173-235.
10. F. Gadducci and U. Montanari, *Comparing Logics for Rewriting: Rewriting Logic, Action Calculi and Tile Logic*, TCS, to appear. Available at http://www.di.unipi.it/ũgo/ABSTRACT.html#TCS-Asilomar.
11. R. Heckel, *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems*, Ph.D. thesis, Technical University of Berlin, 1998.
12. M. Korff, *True concurrency semantics for single pushout graph transformations with applications to actor systems*, Information Systems - Correctness and Reusability, World Scientific, 1995, pp. 33–50.
13. M. Korff, *Generalized graph structures with application to concurrent object-oriented systems*, Ph.D. thesis, Technical University of Berlin, 1995.
14. M. Löwe, *Algebraic approach to single-pushout graph transformation*, Theoretical Computer Science, Vol. 109, 1993, 181-224.
15. J. Meseguer and U. Montanari. *Mapping Tile Logic into Rewriting Logic*, Springer, LNCS 1376, 1998, pp. 62–91.
16. U. Montanari, M. Pistore and F. Rossi, *Modeling concurrent, mobile and coordinated systems via graph transformations*, The Handbook of Graph Grammars, vol. 3: Concurrency, Parallelism and Distribution, World Scientific, 1999, pp. 189–268.
17. U. Montanari and F. Rossi, *Graph Rewriting, Constraint Solving and Tiles for Coordinating Distributed Systems*, Applied Categorical Structures 7(4): 333-370; Dec 1999.
18. G. Rozenberg (editor), *The Handbook of Graph Grammars, vol. 1: Foundations*, World Scientific, 1997.
19. G. Taentzer, *Parallel and distributed graph transformation: Formal description and application to communication-based systems*, Ph.D. thesis, Technical University of Berlin, 1996.