Concatenable Graph Processes: Relating Processes and Derivation Traces*

Paolo Baldan, Andrea Corradini, and Ugo Montanari

Dipartimento di Informatica - University of Pisa
Corso Italia, 40, 56125 Pisa, Italy
E-mail: {baldan, andrea, ugo}@di.unipi.it

Abstract. Several formal concurrent semantics have been proposed for graph rewriting, a powerful formalism for the specification of concurrent and distributed systems which generalizes P/T Petri nets. In this paper we relate two such semantics recently proposed for the algebraic double-pushout approach to graph rewriting, namely the *derivation trace* and the *graph process* semantics. The notion of *concatenable graph process* is introduced and then the category of concatenable derivation traces is shown to be isomorphic to the category of concatenable graph processes. As an outcome we obtain a more intuitive characterization of events and configurations of the event structure associated to a graph grammar.

1 Introduction

The *theory of graph grammars* (or of *graph rewriting systems*) studies a variety of formalisms which extend the theory of formal languages in order to deal with structures more general than strings, like graphs and maps. A graph grammar allows one to describe finitely a (possibly infinite) collection of graphs, i.e., those graphs which can be obtained from a start graph through repeated applications of graph productions. The handbook [15] presents a comprehensive introduction to the theory of the most important approaches to graph rewriting.

Since many (natural or artificial) distributed structures can be represented (at a suitable level of abstraction) by graphs, and graph productions act on those graphs with local transformations, it is quite obvious that graph rewriting systems are potentially interesting for the study of the concurrent transformation of structures. In particular, it has been stressed that Petri nets can be regarded as graph rewriting systems that act on a restricted kind of graphs, namely discrete, labelled graphs (that can be considered as sets of tokens labelled by places). Conversely, graph rewriting systems generalize Petri nets not only because they allow for arbitrary (also non-discrete) graphs, but also because they allow for the specification of context-dependent operations, where part of the state is read but not consumed.

* Research partly supported by the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems) and by the EC Esprit WG APPLIGRAPH (Applications of Graph Transformation).

In recent years, various concurrent semantics for graph rewriting systems have been proposed in the literature, some of which are based on the correspondence with Petri nets (see [2]). The aim of this paper is to relate two such semantics introduced recently by the last two authors in joint works with F. Rossi, H. Ehrig and M. Löwe: the *category of concatenable derivation traces* of [5], used there to obtain an event structure semantics, and the *graph processes* proposed in [6]. Both semantics are worked out, in the mentioned papers, for the algebraic, double-pushout approach to graph transformation [9, 7]. *Derivation traces* are equivalence classes of derivations with respect to two equivalences: a suitable refinement of the isomorphism relation (which makes use of *standard isomorphisms* to guarantee the concatenability of traces); and the *shift* equivalence, relating derivations that differ only for the order in which independent direct derivations are performed. Thus the concurrent semantics is obtained by collecting in equivalence classes all derivations that are conceptually indistinguishable. *Graph processes* are for graph grammars what deterministic, non-sequential processes [10] are for P/T Petri nets. A graph process of a graph grammar $\mathcal{G}$ is an "occurrence grammar" $\mathcal{O}$, i.e., a grammar satisfying suitable aciclicity constraints, equipped with a mapping from $\mathcal{O}$ to $\mathcal{G}$. This mapping is used to associate to the derivations in $\mathcal{O}$ corresponding derivations in $\mathcal{G}$, which can be shown to be *shift*-equivalent. Therefore a process can be regarded as an abstract representation of a class of *shift*-equivalent derivations, starting from the start graph of a grammar: as such it plays a rôle similar to *canonical derivations* [11].

Derivation traces and processes are rather different entities. In fact, while a trace is an equivalence class of derivations, a process is a single structure representing explicitly the dependencies between productions. Moreover, no operation of concatenation has been defined on processes, while derivation traces are concatenable, as they are the arrows of a category. Finally, processes have been defined for *typed* grammars, i.e., grammars where all graphs are endowed with a "typing morphism" to a *type graph* which is fixed for each grammar. This slight variation of the basic definition of graph grammars is used in a critical way in [6]. Derivation traces, instead, have been defined for classical, non-typed grammars. The paper provides a bridge between processes and traces by introducing *concatenable graph processes*, which enrich processes with some additional information needed to be able to concatenate them, and showing that they are in bijective correspondence with *concatenable linear derivation traces*, a slight variation of the traces of [5].

The paper is structured as follows. Section 2 introduces the basics of *typed graph grammars* following the double pushout approach, and defines the category of *concatenable linear derivation traces*. Even if this definition differs in many aspects from the definition of the category of derivation traces in [5], the two categories turn out to be isomorphic. After recalling in Section 3 the basics of graph processes as proposed in [6], Section 4 introduces the key notion of *concatenable graph process* and the corresponding category. Section 5 presents the main result of the paper, i.e., the fact that the category of concatenable linear derivation traces is isomorphic to the category of concatenable processes. In

Section 6, exploiting the main result, we present a quite intuitive characterization of the configurations and events of the event structure of a grammar, as defined in [5], in terms of suitable classes of processes. Finally, Section 7 suggests some further directions of investigation.

2 Typed Graph Grammars

In this section we review the basic definitions about typed graph grammars as introduced in [6], following the algebraic approach based on double pushout construction [9]. Then a category $\mathbf{LTr}[\mathcal{G}]$ of concatenable linear derivation traces of a grammar $\mathcal{G}$ is introduced, by reformulating, in the typed framework, some notions of [5].

Recall that a *(directed, unlabelled) graph* is a tuple $G = \langle N, E, s, t \rangle$, where N is a finite set of *nodes*, E is a finite set of *arcs*, and $s, t : E \to N$ are the *source* and *target* functions. Sometimes we will denote by N_G and E_G the set of nodes and arcs of a graph G. A *graph morphism* $f : G \to G'$ is a pair of functions $f = \langle f_N : N \to N', f_E : E \to E' \rangle$ such that $f_N \circ s = s' \circ f_E$ and $f_N \circ t = t' \circ f_E$; it is an *isomorphism* if both f_N and f_E are bijections; moreover, an *abstract graph* $[G]$ is an isomorphism class of graphs, i.e., $[G] = \{H \mid H \simeq G\}$. An *automorphism* of G is an isomorphism $h : G \to G$. The category having graphs as objects and graph morphisms as arrows is called **Graph**.

A typed graph, as introduced in [6], is a graph G whose nodes and arcs are labelled (by a morphism) over elements of a structure TG that is itself a graph.

Definition 1 (typed graphs category). The category **TG-Graph** of graphs typed over a graph of types TG, is the comma category ($\mathbf{Graph} \downarrow TG$). Objects of **TG-Graph** are denoted $\langle G, t_G \rangle$, where G is a graph and $t_G : G \to TG$ is a graph morphism. □

Definition 2 (typed graph grammar). A *(TG-typed graph) production* is a span $(L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ of injective typed graph morphisms. It is called *consuming* if morphism $l : K \to L$ is not an isomorphism. The typed graphs L, K, and R are called the *left-hand side*, the *interface*, and the *right-hand side* of the production.

A *(TG-typed) graph grammar* $\mathcal{G}$ is a tuple $\langle TG, G_s, P, \pi \rangle$, where G_s is the *start (typed) graph*, P is a set of *production names*, and π maps each production name in P into a graph production. Sometimes we write $q : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ for $\pi(q) = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$. Grammar $\mathcal{G}$ is called *consuming* if all its productions are consuming. □

Since in this paper we work only with typed notions, when clear from the context we omit the word "typed" and do not indicate explicitly the typing morphisms. Moreover, we will consider only *consuming* grammars. This corresponds to the requirement of having non-empty preconditions in the case of Petri nets.

Definition 3 ((linear) direct derivation). Given a typed graph G, a production $q : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$, and an *occurrence* (i.e., a typed graph morphism)

$g : L \to G$, a *(linear) direct derivation* δ *from* G *to* H *using* q *(based on* g*)* exists if and only if the diagram below can be constructed, where both squares are required to be pushouts in **TG-Graph**.

$$\begin{tikzcd} q: L \arrow[d, "g"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] & R \arrow[d, "h"] \\ G & D \arrow[l, "b"] \arrow[r, "d"'] & H \end{tikzcd}$$

In this case, D is called the *context* graph, and we write either $\delta : G \Rightarrow_q H$ or $\delta : G \overset{\langle g,k,h,b,d\rangle}{\Longrightarrow}_q H$, indicating explicitly all the involved morphisms. Since pushouts are defined only up to isomorphism, given isomorphisms $\kappa : G' \to G$ and $\nu : H \to H'$, also $G' \overset{\langle \kappa^{-1} \circ g,k,h,\kappa^{-1} \circ b,d\rangle}{\Longrightarrow}_q H$ and $G \overset{\langle g,k,h\circ\nu,b,d\circ\nu\rangle}{\Longrightarrow}_q H'$ are direct derivations, that we denote respectively by $\kappa \cdot \delta$ and $\delta \cdot \nu$. □

Informally, the rewriting step removes (the image of) the left-hand side from the graph G and substitutes it by (the image of) the right-hand side R. The interface K (common part of L and R) specifies what is preserved.

Definition 4 ((linear) derivations). A *(linear) derivation* over $\mathcal{G}$ is a sequence of (linear) direct derivations (over $\mathcal{G}$) $\rho = \{G_{i-1} \Rightarrow_{q_{i-1}} G_i\}_{i \in \underline{n}}$, where $\underline{n}$ denotes the set of natural numbers $\{1, \ldots, n\}$. The derivation is written $\rho : G_0 \Rightarrow^*_{\mathcal{G}} G_n$ or simply $\rho : G_0 \Rightarrow^* G_n$. The graphs G_0 and G_n are called the *starting* and the *ending graph* of ρ, and are denoted by $\sigma(\rho)$ and $\tau(\rho)$, respectively. The derivation consisting of a single graph G (with $n = 0$) is called the *identity derivation* on G. The *length* $|\rho|$ of ρ is the number of direct derivations in ρ. Given two derivations ρ and ρ' such that $\tau(\rho) = \sigma(\rho')$, we define the *concrete* sequential composition $\rho\,;\rho' : \sigma(\rho) \Rightarrow^* \tau(\rho')$, as the derivation obtained by identifying $\tau(\rho)$ with $\sigma(\rho')$. □

If $\rho : G \Rightarrow H$ is a linear derivation, with $|\rho| > 0$, and $\kappa : G' \to G$, $\nu : H \to H'$ are graph isomorphisms, then $\kappa \cdot \rho : G' \Rightarrow H$ and $\rho \cdot \nu : G \Rightarrow H'$ are defined in the expected way.

In the theory of the algebraic approach to graph grammars, it is natural to reason in terms of *abstract graphs* and *abstract derivations*, considering as equivalent graphs or derivations, respectively, which only differ for representation dependent details. However the definition of abstract derivations is a non-trivial task, if one wants to have a meaningful notion of sequential composition on such derivations. Roughly speaking, the difficulty is represented by the fact that two isomorphic graphs are, in general, related by more than one isomorphism, but to concatenate derivations keeping track of the flow of causality one must specify how the items of two isomorphic graphs should be identified. For an extensive treatment of this problem we refer the reader to [4, 3], which propose a solution based on the choice of a uniquely determined isomorphism, named *standard isomorphism*, relating each pair of isomorphic graphs. Here we follow a slightly different technique: Inspired by the theory of Petri nets, and in particular by the notion of concatenable net process [8], and borrowing a technique proposed in [12], we choose for each class of isomorphic typed graphs a specific graph, named *canonical graph*, and we decorate the starting and ending graphs of a derivation with a pair of isomorphisms from the corresponding canonical graphs to such graphs. In such a way we are allowed to distinguish "equivalent"[1] elements in the starting and ending graphs of derivations and we can safely define their sequential composition.

Definition 5 (canonical graphs). We denote by Can the operation that associates to each (TG-typed) graph its *canonical graph*, satisfying:

1. $Can(G) \simeq G$;
2. if $G \simeq G'$ then $Can(G) = Can(G')$.

□

The construction of the canonical graph can be performed by adapting to our slightly different framework the ideas of [12].

Definition 6 (decorated derivation). A *decorated derivation* $\psi : G_0 \Rightarrow^* G_n$ is a triple $\langle m, \rho, M \rangle$, where $\rho : G_0 \Rightarrow^* G_n$ is a derivation and $m : Can(G_0) \to G_0$, $M : Can(G_n) \to G_n$ are isomorphisms. We define $\sigma(\psi) = Can(\sigma(\rho))$, $\tau(\psi) = Can(\tau(\rho))$ and $|\psi| = |\rho|$. The derivation is called *proper* if $|\psi| > 0$. □

Definition 7 (sequential composition). Let $\psi = \langle m, \rho, M \rangle$, $\psi' = \langle m', \rho', M' \rangle$ be two decorated derivations such that $\tau(\psi) = \sigma(\psi')$. Their *sequential composition* $\psi; \psi'$, is defined, if ψ and ψ' are proper, as $\langle m, \rho \cdot M^{-1}; m' \cdot \rho', M' \rangle$. Otherwise, if $|\psi| = 0$ then $\psi; \psi' = \langle m' \circ M^{-1} \circ m, \rho', M' \rangle$, and similarly, if $|\psi'| = 0$ then $\psi; \psi' = \langle m, \rho, M \circ m' \circ M^{-1} \rangle$. □

The abstraction equivalence identifies derivations that differ only in representation details. As for $\equiv^{sh}$ and $\equiv^{c}$, introduced in the following, such equivalence is a reformulation, in our setting, of the equivalences defined in [5].

Definition 8 (abstraction equivalence). Let $\psi = \langle m, \rho, M \rangle$, $\psi' = \langle m', \rho', M' \rangle$ be two decorated derivations, with $\rho : G_0 \Rightarrow^* G_n$ and $\rho' : G'_0 \Rightarrow^* G'_{n'}$ (whose i^{th} step is depicted in the low rows of Fig. 1). Then they are *abstraction equivalent* if $n = n'$, $q_{i-1} = q'_{i-1}$ for all $i \in \underline{n}$, and there exists a family of isomorphisms $\{\theta_{X_i} : X_i \to X'_i \mid X \in \{G, D\}, i \in \underline{n}\} \cup \{\theta_{G_0}\}$, between corresponding graphs in the two derivations, such that

1. the isomorphisms relating the starting and ending graphs commute with the decorations, i.e. $\theta_{G_0} \circ m = m'$ and $\theta_{G_n} \circ M = M'$;
2. the resulting diagram (step i is represented in Fig. 1) commutes.

Equivalence classes of decorated derivations w.r.t. $\equiv^{abs}$ are called *abstract derivations* and are denoted by $[\psi]_{abs}$, where ψ is an element of the class. □

From a concurrent perspective, two derivations which only differ for the order in which two independent direct derivations are applied, should not be distinguished. This is formalized by the classical shift equivalence on derivations.

---

[1] With "equivalent" we mean here two items related by an automorphism of the graph, that are, in absence of further informations, indistinguishable.

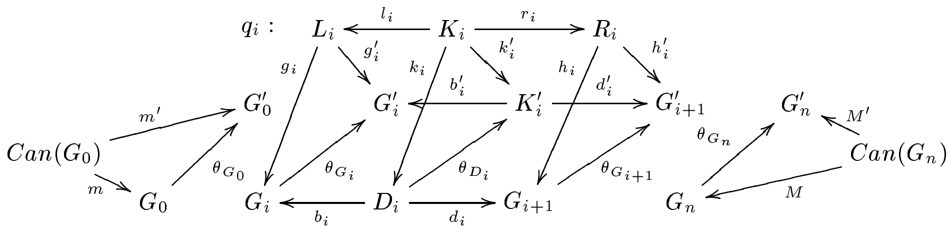

Fig. 1. Abstraction equivalence of decorated derivations.

Definition 9 (shift equivalence). Two direct derivations $\delta_1 : G \Rightarrow_{q_1,g_1} X$ and $\delta_2 : X \Rightarrow_{q_2,g_2} H$ (as in figure below) are *sequentially independent* if $g_2(L_2) \cap h_1(R_1) \subseteq g_2(l_2(K_2)) \cap h_1(r_1(K_1))$; in words, if the left-hand side of q_2 and the right-hand side of q_1 overlap only on items that are preserved by both steps.

$$\begin{tikzcd}
 & q_1 : & & & & q_2 : & \\
L_1 \arrow[d, "g_1"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d, "k_1"'] & R_1 \arrow[dr, "h_1"] & & L_2 \arrow[dl, "g_2"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_1 \arrow[d, "h_2"] \\
G & D_1 \arrow[l, "b_1"] \arrow[rr, "d_1"'] & & X & & D_2 \arrow[ll, "b_2"] \arrow[r, "d_2"'] & H
\end{tikzcd}$$

Given a derivation $\rho = G \Rightarrow_{q_1,g_1} X \Rightarrow_{q_2,g_2} H$, consisting of two sequentially independent direct derivations, there is a constructive way to obtain a new derivation $\rho' : G \Rightarrow_{q_2,g_2'} X' \Rightarrow_{q_1,g_1'} H$, where productions q_1 and q_2 are applied in the reverse order. We say that ρ' is a *switching* of ρ and we write $\rho \sim^{sh} \rho'$.
The *shift equivalence* $\equiv^{sh}$ on concrete derivations is the transitive and "context" closure of $\sim^{sh}$, i.e. the least equivalence, containing $\sim^{sh}$, such that if $\rho \equiv^{sh} \rho'$ then $\rho_1; \rho; \rho_3 \equiv^{sh} \rho_1; \rho'; \rho_3$. The same symbol denotes the equivalence on decorated derivations induced by $\equiv^{sh}$, i.e. $\langle m, \rho, M\rangle \equiv^{sh} \langle m, \rho', M\rangle$ if $\rho \equiv^{sh} \rho'$. □

Definition 10 (ctc-equivalence). The *concatenable truly concurrent equivalence* (*ctc-equivalence*) $\equiv^c$ on derivations is the transitive closure of the union of the relations $\equiv^{abs}$ and $\equiv^{sh}$. Equivalence classes of decorated derivations with respect to $\equiv^c$ are denoted as $[\psi]_c$ and are called *concatenable linear (derivation) traces.* □

It is possible to prove that if $\psi \equiv^c \psi'$ and $\psi_1 \equiv^c \psi_1'$ then, if defined, $\psi; \psi_1 \equiv^c \psi'; \psi_1'$. Therefore sequential composition of decorated derivations lifts to composition of linear derivation traces.

Definition 11 (category of concatenable linear traces). The *category of concatenable linear traces* of a grammar $\mathcal{G}$, denoted by $\mathbf{LTr}[\mathcal{G}]$, has abstract graphs as objects and concatenable linear traces as arrows. In particular, if $\psi : G \Rightarrow_{\mathcal{G}}^* H$, then $[\psi]_c$ is an arrow from $[G]$ to $[H]$. The identity arrow on $[G]$ is the ctc-equivalence class of the identity derivation $\langle i, G, i\rangle$, where $i : Can(G) \to G$ is any isomorphism. □

In [5] a category $\mathbf{Tr}[\mathcal{G}]$ of concatenable (parallel) traces is defined considering possibly parallel derivations and using standard isomorphisms instead of decorations. More precisely, a class of standard isomorphisms is fixed and abstraction equivalence on (parallel) derivations is defined as in Definition 8, but replacing condition 1 with the requirement for the isomorphisms θ_0 and θ_n, relating the starting and ending graphs, to be standard. Then the concatenable truly concurrent equivalence on parallel derivations is again defined as the least equivalence containing the abstraction and shift equivalences. Despite of these differences, the two approaches lead to the same category of traces.

Proposition 12. *The category of concatenable parallel traces* $\mathbf{Tr}[\mathcal{G}]$ *and the category* $\mathbf{LTr}[\mathcal{G}]$ *of concatenable linear traces are isomorphic.*

3 Graph Processes

Graph processes, introduced in [6], generalize the notion of (deterministic, non-sequential) process of a P/T net [10] to graph grammars. A graph process of a graph grammar $\mathcal{G}$ is an "occurrence grammar" $\mathcal{O}$, i.e., a grammar satisfying suitable acyclicity constraints, equipped with a mapping from $\mathcal{O}$ to $\mathcal{G}$.

Definition 13 (strongly safe grammar). A *strongly safe* graph grammar is a grammar $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ such that each graph H reachable from the start graph (i.e., $G_s \Rightarrow^* H$) has an injective typing morphism. We denote with $\mathit{Elem}(\mathcal{G})$ the set $N_{TG} \cup E_{TG} \cup P$. $\square$

Without loss of generality, injective morphisms can be seen as inclusions. Thus sometimes a graph $\langle G, m \rangle$, reachable in a strongly safe grammar, will be identified with the subgraph $m(G)$ of TG. In the following, by L_q (resp. K_q, R_q) we will sometimes denote the graph L (resp., K, R) of a production $q : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$. When interested in the typing we assume $L_q = \langle LG_q, tl_q \rangle$, $K_q = \langle KG_q, tk_q \rangle$ and $R_q = \langle RG_q, tr_q \rangle$.

Definition 14 (causal relation). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a strongly safe grammar, let $q \in P$ be a production, and let $x \in N_{TG} \cup E_{TG}$ be any arc or node of the type graph TG. We say that q *consumes* x if $x \in tl_q(LG_q - KG_q)$, that q *creates* x if $x \in tr_q(RG_q - KG_q)$ and that q *preserves* x if $x \in tk_q(KG_q)$.[2]

The *causal relation* of $\mathcal{G}$ is given by the structure $\langle \mathit{Elem}(\mathcal{G}), \leq \rangle$, where $\leq$ is the transitive and reflexive closure of the relation $<$ defined by the following clauses: for any node or arc x in TG, and for productions $q_1, q_2 \in P$

1. $x < q_1$ if q_1 consumes x;
2. $q_1 < x$ if q_1 creates x;
3. $q_1 < q_2$ if q_1 creates x and q_2 preserves x, or q_1 preserves x and q_2 consumes x.

$\square$

The first two clauses of the definition of relation $\leq$ are obvious. The third one formalizes the fact that if an item is generated by q_1 and it is preserved by q_2, then q_2 cannot be applied before q_1, and, symmetrically, if an item is preserved by q_1 and consumed by q_2, then q_1 cannot be applied after q_2.

[2] With abuse of notation, in $LG_q - KG_q$ or $RG_q - KG_q$ graphs are considered as sets of nodes and arcs.

Definition 15 (occurrence grammar). An *(deterministic) occurrence grammar* is a strongly safe graph grammar $\mathcal{O} = \langle TG, G_s, P, \pi \rangle$ such that

1. its causal relation $\leq$ is a partial order, and for any $n \in N_{TG}, e \in E_{TG}$ such that $n = s(e)$ or $n = t(e)$, and for any $q \in P$, we have
   - if $q \leq n$, then $q \leq e$ and
   - if $n \leq q$, then $e \leq q$;
2. consider the set Min of minimal elements of $\langle Elem(\mathcal{G}), \leq \rangle$ and $Min(\mathcal{O}) = \langle Min \cap N_{TG}, Min \cap E_{TG}, s_{|_{Min \cap N_{TG}}}, t_{|_{Min \cap N_{TG}}} \rangle$; then $G_s = Min(\mathcal{O})$;
3. for all $q \in P$, q satisfies the identification condition [9], i.e. there is no $x, y \in LG_q$ such that $tl_q(x) = tl_q(y)$ and $y \notin l(KG_q)$.
4. for all $x \in N_{TG} \cup E_{TG}$, x is consumed by at most one production in P, and it is created by at most one production in P.

$\square$

For an occurrence grammar $\mathcal{O}$, denoted by Max the set of maximal elements in $\langle Elem(\mathcal{O}), \leq \rangle$, let $Max(\mathcal{O}) = \langle Max \cap N_{TG}, Max \cap E_{TG}, s_{|_{Max \cap N_{TG}}}, t_{|_{Max \cap N_{TG}}} \rangle$. Note that, since the start graph of an occurrence grammar $\mathcal{O}$ is determined as $Min(\mathcal{O})$, we often do not mention it explicitly.

Definition 16 (reachable sets). Let $\mathcal{O} = \langle TG, P, \pi \rangle$ be an occurrence grammar, and let $\langle P, \leq \rangle$ be the restriction of the causal relation to the productions of $\mathcal{O}$. For any left-closed $P' \subseteq P$,[3] the *reachable set* associated to P' is the set of nodes and arcs $S_{P'} \subseteq N_{TG} \cup E_{TG}$ defined as

$$x \in S_{P'} \quad \text{iff} \quad \forall q \in P \,.\, (x \leq q \Rightarrow q \notin P') \wedge (x \geq q \Rightarrow q \in P').$$

We denote by $G(S_{P'})$ the structure $\langle S_{P'} \cap E_{TG}, S_{P'} \cap N_{TG}, s_{|_{S_{P'} \cap E_{TG}}}, t_{|_{S_{P'} \cap E_{TG}}} \rangle$.
$\square$

For any reachable set $S_{P'}$, $G(S_{P'})$ is a graph and it is reachable from $Min(\mathcal{O})$ with a derivation which applies exactly once every production in P', in any order consistent with $\leq$.

As a consequence, in particular $Min(\mathcal{O}) = G(S_\emptyset)$ and $Max(\mathcal{O}) = G(S_P)$ are well-defined subgraphs of TG and $Min(\mathcal{O}) \Rightarrow^*_P Max(\mathcal{O})$, using all productions in P exactly once, in any order consistent with $\leq$. This makes clear why a graph process of a grammar $\mathcal{G}$, that we are going to define as an occurrence grammar plus a mapping to the original grammar, can be seen as a representative of a set of derivations of $\mathcal{G}$, where only independent steps may be switched.

Definition 17 (process). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar. A *process* p for $\mathcal{G}$ is a pair $\langle \mathcal{O}, \phi \rangle$, where $\mathcal{O} = \langle TG', P', \pi' \rangle$ is an occurrence grammar and $\phi = \langle mg, mp, \iota \rangle$, with

- $mg : TG' \to TG$ a graph morphism;

[3] A subset Y of an ordered set $\langle X, \leq \rangle$ is *left-closed* if for any $x \in X$ and $y \in Y$, $x \leq y$ implies $x \in Y$.

- $mp : P' \to P$ a function mapping each production $q' : (L' \leftarrow K' \to R')$ in P' into an isomorphic production $q = mp(q') : (L \leftarrow K \to R)$ in P and
- ι a function mapping each production $q' \in P'$ into a triple of isomorphisms $\iota(q') = \langle \iota^L(q') : L \to L', \iota^K(q') : K \to K', \iota^R(q') : R \to R' \rangle$, making the diagram in Fig. 2.(a) commute.

We denote with $Min(p)$ and $Max(p)$ the graphs $Min(\mathcal{O})$ and $Max(\mathcal{O})$ typed over TG by the corresponding restrictions of mg. □

Notice that, unlike in [6], processes are not forced to "start" from the start graph of the grammar. This is needed, in the next section, to define a reasonable notion of concatenable process.

$$
\begin{tikzcd}
L' \arrow[dr] & K' \arrow[l] \arrow[r] \arrow[d] & R' \arrow[dl] \\
& TG' \arrow[d, "mg"] & \\
& TG & \\
L \arrow[uuu, "\iota^L(q')"] \arrow[ur] & K \arrow[l] \arrow[r] \arrow[u] \arrow[uuu, bend left, "\iota^K(q')"] & R \arrow[uuu, "\iota^R(q')"'] \arrow[ul]
\end{tikzcd}
\qquad
\begin{tikzcd}
L_1 \arrow[dd, "tl_1"'] & L \arrow[l, "\iota_1^L(p_1)"'] \arrow[r, "\iota_1^L(p_2)"] \arrow[d, "tl"'] & L_2 \arrow[dd, "tl_2"] \\
& TG & \\
TG_1 \arrow[ur, "mg_1"] \arrow[rr, bend right, "fg"'] & & TG_2 \arrow[ul, "mg_2"']
\end{tikzcd}
$$

(a) *(b)*

Fig. 2. Processes and isomorphisms of processes

Definition 18 (isomorphism of processes). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar and let $p_j = \langle \mathcal{O}_j, \phi_j \rangle$, with $\mathcal{O}_j = \langle TG_j, P_j, \pi_j \rangle$ and $\phi_j = \langle mg_j, mp_j, \iota_j \rangle$, for $j = 1, 2$, be two processes of $\mathcal{G}$. An *isomorphism* between p_1 and p_2 is a pair $\langle fg, fp \rangle : p_1 \to p_2$ such that

1. $fg : \langle TG_1, mg_1 \rangle \to \langle TG_2, mg_2 \rangle$ is an isomorphism (of TG-typed graphs);
2. $fp : P_1 \to P_2$ is a bijection s.t. $mp_1 = mp_2 \circ fp$;
3. for each $q_1 : (L_1 \leftarrow K_1 \to R_1)$ in P_1, $q_2 = fp(q_1) : (L_2 \leftarrow K_2 \to R_2)$ in P_2, $q = mp_1(q_1) = mp_2(q_2) : (L \leftarrow K \to R)$ in P, the diagram in Fig. 2.(b) and the analogous ones for the interfaces and the right-hand sides, commute.

If there exists an isomorphism $f : p_1 \to p_2$ we say that p_1 and p_2 are *isomorphic* and we write $p_1 \cong p_2$. □

This definition is slightly more restrictive than the original one in [6], since, guided by the notion of abstraction equivalence for decorated derivations, we require commutativity of diagrams in Fig. 2.(b) w.r.t. to fixed isomorphisms $\iota^L(p_j), \iota^K(p_j), \iota^R(p_j)$, which are part of the processes, and not w.r.t. generic isomorphisms as in [6].

4 Concatenable Processes

Since processes represent (concurrent) computations and express explicitly the causal dependencies existing between single rewriting steps, it is very natural to ask for a notion of sequential composition of processes consistent with causal dependencies. To define such notion we have to face the same problem described for traces and we solve it exactly in the same way, i.e. by decorating the source $Min(p)$ and the target $Max(p)$ of the process p with isomorphisms from the corresponding canonical graphs. Such isomorphisms play the same rôle of the ordering on maximal and minimal places of *concatenable processes* in Petri net theory [8]. In this view our *concatenable* graph processes are related to the graph processes of [6] in the same way as the concatenable processes of [8] are related to the classical Goltz-Reisig processes for P/T nets [10].

The technique of decorating the source and target graphs of a process with graph isomorphisms is essentially the same as that used in [12] to make *dynamic graphs* concatenable. However dynamic graphs are very different from processes. Firstly, unlike processes, dynamic graphs are not defined as (a suitable kind of) graph grammars, which implies, for example, that the analogy with processes as they are defined in Petri nets theory is lost. Secondly, dynamic graphs are studied in [12] as mathematical structures in their own, placing the emphasys on their composition/decomposition properties, while in our approach processes as considered as an abstract description of graph grammar computations.

Definition 19 (concatenable process). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed grammar. A *concatenable process (c-process)* for $\mathcal{G}$ is a triple $cp = \langle m, p, M \rangle$, where p is a process and $m : Can(Min(p)) \to Min(p)$, $M : Can(Max(p)) \to Max(p)$ are isomorphisms (of TG-typed graphs). We denote with $Min(cp)$ and $Max(cp)$ the graphs $Min(p)$ and $Max(p)$.

If the occurrence grammar $\mathcal{O} = \langle TG', P', \pi' \rangle$ associated to the process has an empty set of productions (and thus $Min(p) = Max(p) = TG'$), the c-process cp is called a *symmetry* and denoted as $Sym_{\mathcal{G}}(m, \langle TG', mg \rangle, M)$ or simply as $Sym_{\mathcal{G}}(m, TG', M)$, if omitting the typing morphism is not misleading. □

Definition 20 (isomorphism of c-processes). Let $cp_j = \langle m_j, p_j, M_j \rangle$ for $j = 1, 2$, be two c-processes. An *isomorphism* between cp_1 and cp_2 is an isomorphism of processes $\langle fg, fp \rangle : p_1 \to p_2$ such that the following diagrams commute:

$$\begin{array}{ccc} Can(Min(cp_1)) & \xrightarrow{m_1} & Min(cp_1) \\ \| & & \downarrow fg \\ Can(Min(cp_2)) & \xrightarrow{m_2} & Min(cp_2) \end{array} \qquad \begin{array}{ccc} Max(cp_1) & \xleftarrow{M_1} & Can(Max(cp_1)) \\ \downarrow fg & & \| \\ Max(cp_2) & \xleftarrow[M_2]{} & Can(Max(cp_2)) \end{array}$$

where fg denote the restrictions of fg itself to the corresponding graphs. If there exists an isomorphism $f : cp_1 \to cp_2$ we say that cp_1 and cp_2 are *isomorphic* and we write $cp_1 \cong cp_2$. An isomorphism class of c-processes is called *abstract c-process* and denoted by $[cp]$, where cp is a member of the class. □

Given two c-processes cp_1 and cp_2 such that $Max(cp_1) \simeq Min(cp_2)$, we can concatenate them by gluing the *Max* graph of the first one with the *Min* graph of the second one. Formally, the type graph of the resulting process is obtained via a pushout construction and thus it is defined only up to isomorphism. However, when lifted to the abstract setting the operation turns out to be well-defined and deterministic.

Definition 21 (sequential composition). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar and let $[cp_1]$ and $[cp_2]$ be two abstract c-processes for $\mathcal{G}$ (with $cp_j = \langle m_j, p_j, M_j \rangle$, $p_j = \langle \mathcal{O}_j, \phi_j \rangle = \langle \langle TG_j, P_j, \pi_j \rangle, \langle mg_j, mp_j, \iota_j \rangle \rangle$), such that $Max(cp_1) \simeq Min(cp_2)$. The *sequential composition* of $[cp_1]$ and $[cp_2]$, denoted by $[cp_1];[cp_2]$ is the isomorphism class $[cp]$ of the c-process:

$$cp = \langle m, p, M \rangle,$$

where $p = \langle \mathcal{O}', \phi' \rangle = \langle \langle G'_s, TG', P', \pi' \rangle, \langle mg', mp' \rangle \rangle$, is defined as follows. The type graph TG', with the morphism $mg' : TG' \to TG$, is given by the following pushout diagram (in **TG-Graph**):

$$\begin{tikzcd}
Can(Max(cp_1)) \arrow[r, "M_1"] \arrow[dd, equal] & Max(cp_1) \arrow[r, hook] & TG_1 \arrow[dr, "g_1"] & \\
& & & TG' \\
Can(Max(cp_2)) \arrow[r, "m_2"'] & Max(cp_2) \arrow[r, hook] & TG_2 \arrow[ur, "g_2"'] &
\end{tikzcd}$$

The set of production names is $P' = P_1 \cup P_2$.[4] For $q_j \in P_j$ $(j = 1, 2)$ $\pi'(q_j)$ is obtained typing over TG' the corresponding production $\pi_j(q_j)$, by composing the typing morphism with g_j. The function mp' is defined as $mp'(q_j) = mp_j(q_j)$ for $j = 1, 2$ and $q_j \in P_j$. Finally the restrictions of $g_1 : Min(cp_1) \to Min(cp)$ and $g_2 : Max(cp_2) \to Max(cp)$ are isomorphisms so that we can define $m = g_1 \circ m_1$ and $M' = g_2 \circ M_2$. Finally $G'_s = g_1(Min(cp_1))$. □

Definition 22 (category of (abstract) c-processes). Given a typed graph grammar $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$, we denote by $\mathbf{CP}[\mathcal{G}]$ the category of *(abstract)* c-processes having abstract graphs typed over TG as objects and abstract c-processes as arrows. An abstract c-process $[cp]$ is an arrow from $[Min(cp)]$ to $[Max(cp)]$. The identity on $[G]$ is the symmetry $[Sym(i, G, i)]$ (where $i : Can(G) \to G$ is any isomorphism). □

5 Relating traces and processes

This section shows that the semantic model based on concatenable linear traces and the one based on concatenable graph processes are essentially the same. More formally we prove that the category $\mathbf{LTr}[\mathcal{G}]$ of concatenable linear traces (Definition 11) is isomorphic to the category of abstract c-processes $\mathbf{CP}[\mathcal{G}]$.

First, given an abstract c-process $[cp]$ we can obtain a derivation by "executing" the productions of cp in any order compatible with the causal order.

Definition 23 (from processes to traces). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar and let $[cp]$ be an abstract c-process of $\mathcal{G}$, where $cp = \langle m, p, M \rangle$, $p = \langle \mathcal{O}, \phi \rangle = \langle \langle TG', P', \pi' \rangle, \langle mg, mp, \iota \rangle \rangle$. Let $q'_0, \ldots, q'_{n-1}$ be an enumeration of the productions of P' compatible with the causal order of cp.

We associate to $[cp]$ the concatenable linear trace $\mathcal{L}_A([cp]) = [\psi]_c$, with

[4] The sets P_1 and P_2 are supposed to be disjoint.

$$\psi = \langle m, \rho, M \rangle, \quad \text{where } \rho = \{G_{i-1} \Rightarrow_{q_{i-1}, g_{i-1}} G_i\}_{i \in \underline{n}}$$

such that $G_0 = Min(cp)$, $G_n = Max(cp)$, and for each $i = 0, \dots, n-1$

- $q_i = mp(q'_i)$;
- $G_{i+1} = G(S_{\{q'_0, \dots, q'_i\}})$, i.e. G_{i+1} is the subgraph of the type graph TG' of the process determined by the reachable set $S_{\{q'_0, \dots, q'_i\}}$, typed by mg;
- each derivation step $G_i \Rightarrow_{q_i, g_i} G_{i+1}$ is as in Fig. 3.(a), where unlabelled arrows represent inclusions.

□

$$\begin{tikzcd}
q_i = mp(q'_i) : LG_i \arrow[d, "\iota^L(q'_i)"'] & KG_i \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "\iota^K(q'_i)"'] & RG_i \arrow[d, "\iota^R(q'_i)"'] \\
q'_i : LG'_i \arrow[d, "tl_i"'] & KG'_i \arrow[l, "l'_i"] \arrow[r, "r'_i"'] \arrow[d, "tk_i"'] & RG_i \arrow[d, "tr_i"'] \\
G_i & D_i \arrow[l] \arrow[r] & G_{i+1}
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
q_i : L_i \arrow[d, "g_i"'] & K_i \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "k_i"'] & R_i \arrow[d, "h_i"] \\
G_i \arrow[dr, "cg_i"'] & D_i \arrow[l, "b_i"'] \arrow[r, "d_i"] \arrow[d, "cd_i"] & G_{i+1} \arrow[dl, "cg_{i+1}"] \\
& TG' &
\end{tikzcd}$$

(b)

Fig. 3. From abstract c-processes to concatenable linear traces and backward.

It can be shown that ψ is a legal decorated derivation in $\mathcal{G}$ and its equivalence class $[\psi]_c$ does not depend on the particular choice of the representative cp, thus the mapping $\mathcal{L}_A$ is well defined. Moreover $\mathcal{L}_A$ preserves sequential composition of processes and identities, and thus it can be lifted to a functor $\mathcal{L}_A : \mathbf{CP}[\mathcal{G}] \to \mathbf{LTr}[\mathcal{G}]$ which acts as identity on objects.

The backward step, from concatenable linear traces to abstract c-processes, is performed via a colimit construction that, applied to a derivation in the trace, essentially constructs the type graph as a copy of the starting graph plus the items produced during the rewriting process. Productions of the process are instances of production applications.

Definition 24 (from traces to processes). Let $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ be a typed graph grammar and let $[\psi]_c$ be a concatenable linear trace, with $\psi = \langle m, \rho, M \rangle$. We associate to $[\psi]_c$ an abstract c-process $\mathcal{P}_A([\psi]_c) = [cp]$, with $cp = \langle m', p, M' \rangle$, $p = \langle \mathcal{O}, \phi \rangle = \langle \langle TG', P', \pi' \rangle, \langle mg, mp, \iota \rangle \rangle$, such that:

- $\langle TG', mg \rangle$ is the colimit object (in category **TG-Graph**) of the diagram representing derivation ψ, as depicted (for a single derivation step and without typing morphisms) in Fig. 3.(b);
- $P' = \{\langle q_i, i \rangle \mid q_i \text{ is used in step } i, \text{ for } i = 0, \dots, n-1\}$, and for all $i = 0, \dots, n-1$, referring to Fig. 3.(b), $\pi'(\langle q_i, i \rangle) = (\langle LG_i, cg_i \circ g_i \rangle \stackrel{l_i}{\leftarrow} \langle KG_i, cd_i \circ k_i \rangle \stackrel{r_i}{\rightarrow} \langle RG_i, cg_{i+1} \circ h_i \rangle)$, $mp(\langle q_i, i \rangle) = q_i$ and $\iota(\langle q_i, i \rangle) = \langle id_{L_i}, id_{K_i}, id_{R_i} \rangle$. Note that for $i \neq j$ we may have $q_i = q_j$; instead, the productions in P' are all distinct, as they can be considered as production occurrences of $\mathcal{G}$;
- $m' = cg_0 \circ m$ and $M' = cg_n \circ M$;

□

It is possible to prove that $\mathcal{P}_A : \mathbf{Abs}[\mathcal{G}] \to \mathbf{LCP}[\mathcal{G}]$, obtained extending $\mathcal{P}_A$ as identity on objects, is a well defined functor, and that $\mathcal{L}_A$ and $\mathcal{P}_A$ are inverse each other.

Theorem 25. *Let $\mathcal{G}$ be a graph grammar. Then $\mathcal{L}_A : \mathbf{CP}[\mathcal{G}] \to \mathbf{LTr}[\mathcal{G}]$ and $\mathcal{P}_A : \mathbf{LTr}[\mathcal{G}] \to \mathbf{CP}[\mathcal{G}]$ are inverse each other, establishing an isomorphism of categories.*

6 Processes and events

Event structures [17] are a simple event based model of concurrent computations in which events are considered as atomic, indivisible and instantaneous steps, which can appear only once in a computation. An event can occur only after some other events have taken place and the execution of some events can inhibit the execution of other events. We consider here *prime* event structures with *binary conflict* (PES). A PES consists of a set of events endowed with two binary relations: *causality*, modelled by a partial order relation $\leq$, and *conflict*, modeled by a symmetric and irreflexive relation $\#$, hereditary w.r.t. causality. A *configuration* of a PES is a left-closed, conflict free subset of events, representing a possible computation of the system modelled by the event structure. The set of configurations of a PES, ordered by subset inclusion, is a finitary prime algebraic domain, i.e. a coherent, prime algebraic, finitary partial order, briefly a *domain*, and the set of prime elements of a domain (with the induced partial order as causality and the inconsistency relation as conflict) is a PES.

The category of concatenable traces $\mathbf{Tr}[\mathcal{G}]$ is used in [5] to define the domain and the event structure of a grammar $\mathcal{G}$. Elements of the domain are suitable classes of concatenable traces. Proposition 12 implies that the same structure can be obtained starting from category $\mathbf{LTr}[\mathcal{G}]$.

Theorem 26. *For any graph grammar $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ the comma category $[G_s] \downarrow \mathbf{LTr}[\mathcal{G}]$ is a preorder $\mathbf{PreDom}[\mathcal{G}]$, i.e., there is at most one arrow between any pair of objects. Moreover the ideal completion of $\mathbf{PreDom}[\mathcal{G}]$ is a domain, denoted by $\mathbf{Dom}[\mathcal{G}]$.*

By results in [17], $\mathbf{Dom}[\mathcal{G}]$ is the domain of configurations of a uniquely determined PES $\mathbf{ES}[\mathcal{G}]$, which is proposed as the truly concurrent semantics of the grammar. In [5] also an explicit construction is provided for such an event structure, using traces. Here, thanks to the very close relation existing between concatenable processes and concatenable linear traces, stated in Theorem 25, we are able to provide a nice characterization of the finite configurations (finite elements of the domain $\mathbf{Dom}[\mathcal{G}]$) and of the events of $\mathbf{ES}[\mathcal{G}]$. The result resembles the analogous correspondence existing for P/T nets and is based on a similar notion of left concatenable process.

Definition 27 (abstract left c-process). Two c-processes cp_1 and cp_2 are *left isomorphic*, denoted by $cp_1 \equiv_l cp_2$, if there exists a pair of functions $f = \langle fg, fp \rangle$ satisfying all the requirements of Definition 18, but, possibly, the commutativity of the right triangle of Fig. 2. An *abstract left c-process* is a class of left isomorphic c-processes $[cp]_l$. It is *initial* if $Min(cp) \simeq G_s$. It is *prime* if the causal order $\leq$ of cp, restricted to the set P of its productions, has a maximum element. □

The following result has a clear intuitive meaning if one think of the productions of (the occurrence grammar of) a process as instances of production applications in the original grammar $\mathcal{G}$, and therefore as possible events in $\mathcal{G}$.

Theorem 28. *There is a one to one correspondence between:*

1. *initial left c-processes and finite elements of* $\mathbf{Dom}[\mathcal{G}]$*;*
2. *prime initial left c-processes and elements of* $\mathbf{ES}[\mathcal{G}]$.

This result confirms the appropriateness of the chosen notion of (concatenable) process and makes us confident on the possibility of carrying on the program of extending notions and results from the theory of Petri nets to graph transformation systems.

7 Conclusions

As recalled in the introduction, typed graph grammars can be seen as a proper generalization of P/T Petri nets and many concepts and results in the theory of concurrency for graph grammars manifest an evident similarity with corresponding notions for nets. The deepening and formalization of this analogy represents a direction for future research. In particular, we intend to continue the investigation of the relationship among the various notions of graph and net processes. Furthermore we are trying to extend to graph grammars the unfolding construction of [17, 13] (which generates the event structure associated to a net via the unfolded occurrence net) following, for what concern the handling of asymmetric conflicts, the ideas presented in [1]. Preliminary considerations suggest that graph processes of [6] are in precise correspondence with Goltz-Reisig processes [10]. On the other hand, our concatenable graph processes are not the exact counterpart of the concatenable processes of [8]. This is due to the fact that we have been mainly guided by the aim of unifying the various existing semantics for graph grammars: the equivalence with [5] has been formally proved in this paper and we are confident that a similar result can be obtained for the semantics proposed by Schied in [16]. Furthermore many variations of concatenable processes in the theory of nets exists, enjoying different properties. For instance, the *decorated processes* [14] generate the same domain produced by the unfolding construction. We are convinced that our concatenable graph processes correspond to a slight refinement of such net processes and, therefore, that the equivalence result between the process and unfolding semantics can be extended to the graph rewriting setting.

References

1. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. *FoSSaCS '98*, *LNCS* 1378, pp. 63–80. Springer, 1998.
2. A. Corradini. Concurrent Graph and Term Graph Rewriting. *CONCUR'96*, *LNCS* 1119, pp. 438–464. Springer, 1996.
3. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract Graph Derivations in the Double-Pushout Approach. *Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, *LNCS* 776, pp. 86–103. Springer, 1994.
4. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Note on standard representation of graphs and graph derivations. *Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, *LNCS* 776, pp. 104–118. Springer, 1994.
5. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An Event Structure Semantics for Graph Grammars with Parallel Productions. *5th International Workshop on Graph Grammars and their Application to Computer Science*, *LNCS* 1073. Springer, 1996.
6. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
7. 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 [15].
8. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Informatica*, 33:641–647, 1996.
9. H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. *3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, *LNCS* 291, pp. 3–14. Springer, 1987.
10. U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
11. H.-J. Kreowski. *Manipulation von Graphmanipulationen*. PhD thesis, Technische Universität Berlin, 1977.
12. A. Maggiolo-Schettini and J. Winkowski. Dynamic Graphs. In *MFCS'96*, *LNCS* 1113, pp. 431–442, 1996.
13. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In *CONCUR '92*, *LNCS* 630, pp. 286–301. Springer, 1992.
14. J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for Place/Transition Petri nets. *Theoret. Comput. Sci.*, 153(1-2):171–210, 1996.
15. G. Rozemberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations*. World Scientific, 1997.
16. G. Schied. On relating Rewriting Systems and Graph Grammars to Event Structures. *Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, *LNCS* 776, pp. 326–340. Springer, 1994.
17. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, *LNCS* 255, pp. 325–392. Springer, 1987.