Unfolding and Event Structure Semantics for Graph Grammars*

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. We propose an unfolding semantics for graph transformation systems in the double-pushout (DPO) approach. Mimicking Winskel's construction for Petri nets, a graph grammar is unfolded into an acyclic branching structure, that is itself a (nondeterministic occurrence) graph grammar describing all the possible computations of the original grammar. The unfolding can be abstracted naturally to a prime algebraic domain and then to an event structure semantics. We show that such event structure coincides both with the one defined by Corradini et al. [3] via a comma category construction on the category of concatenable derivation traces, and with the one proposed by Schied [13], based on a deterministic variant of the DPO approach. This results, besides confirming the appropriateness of our unfolding construction, unify the various event structure semantics for the DPO approach to graph transformation.

1 Introduction

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 transformation systems are potentially interesting for the study of the concurrent transformation of structures. In particular, Petri nets [11], the first formal tool proposed for the specification of the behaviour of concurrent systems, can be regarded as graph transformation systems that act on a restricted kind of graphs, namely discrete, labelled graphs (to be interpreted as sets of tokens labelled by place names).

In recent years, various concurrent semantics for graph rewriting systems have been proposed in the literature, some of which are inspired by the mentioned correspondence with Petri nets (see [2] for a tutorial introduction to the topic and for relevant references). A classical result in the theory of concurrency for Petri nets, due to Winskel [15], shows that the event structure semantics of *safe* nets can be given via a chain of adjunctions starting from the category **Safe** of safe nets, through category **Occ** of occurrence nets (this result has been generalized to arbitrary P/T nets in [9]). In particular, the event structure associated with

* Research partially supported by MURST project Tecniche Formali per Sistemi Software, by TMR Network GETGRATS and by Esprit WG APPLIGRAPH.

a net is obtained by first constructing a "nondeterministic unfolding" of the net, and then by extracting from it the events (which correspond to the transitions) and the causal and conflict relations among them.

In the present paper we propose a similar unfolding construction for DPO graph grammars, which can be considered as a first contribution to a functorial semantics in Winskel's style. After recalling in Section 2 the basics of typed graph transformation systems and their correspondence with Petri nets, we introduce, in Section 3, the notion of *nondeterministic occurrence grammar*, a generalization of the deterministic occurrence grammars of [4], representing in a unique "branching" structure several possible "acyclic" computations. Interestingly, unlike the case of Petri nets, the relationships among productions of an occurrence graph grammar cannot be captured completely by two binary relations representing causality and symmetric conflict. Firstly, due to the possibility of preserving some items in a rewriting step an asymmetric notion of conflict has to be considered. The way we face the problem is borrowed from [1], where we addressed an analogous situation arising in the treatment of *contextual nets*. Secondly, further dependencies among productions are induced by the application conditions, which constrain the applicability of the rewrite rules in order to preserve the consistency of the graphical structure of the state.

Next in Section 4 we present an unfolding construction that, when applied to a given grammar $\mathcal{G}$, yields a nondeterministic occurrence grammar $\mathcal{U}_{\mathcal{G}}$, which describes its behaviour. The idea consists of starting from the initial graph of the grammar, applying in all possible ways its productions, and recording in the unfolding each occurrence of production and each new graph item generated by the rewriting process. Our unfolding construction is conceptually similar to the unfolding semantics proposed for graph rewriting in the *single-pushout approach* by Ribeiro in [12]. However, here the situation is more involved and the two approaches are not directly comparable, due to the absence of the application conditions (dangling and identification) in the single-pushout approach.

In Section 5 we show how a prime algebraic domain (and therefore a prime event structure) can be extracted naturally from a nondeterministic occurrence grammar. Then the event structure semantics $ES(\mathcal{G})$ of a grammar $\mathcal{G}$ is defined as the event structure associated to its unfolding $\mathcal{U}_{\mathcal{G}}$. In Section 6 such semantics is shown to coincide with two other event structure semantics for graph rewriting in the literature: the one by Corradini et al. [3], built on top of the *abstract, truly concurrent model of computation* of a grammar (a category having abstract graphs as objects and concatenable derivation traces as arrows), and the one by Schied [13], based on a deterministic variation of the DPO approach. Finally, in Section 7 we conclude and present some possible directions of future work.

2 Typed Graph Grammars

This section briefly summarizes the basic definitions about typed graph grammars [4], a variation of classical DPO graph grammars [6, 5] where the rewriting takes place on the so-called *typed graphs*, namely graphs labelled over a structure (the *graph of types*) that is itself a graph. Besides being strictly more general than usual labelled graphs, typed graphs will also allow us to have a clearer correspondence between graph grammars and Petri nets.

Formally, a *(directed, unlabelled) graph* is a tuple $G = \langle N, E, s, t\rangle$, where N is a set of *nodes*, E is a set of *arcs*, and $s, t : E \to N$ are the *source* and *target* functions. A *graph morphism* $f : G \to G'$ is a pair of functions $f = \langle f_N : N \to N', f_E : E \to E'\rangle$ preserving sources and targets, i.e., such that $f_N \circ s = s' \circ f_E$ and $f_N \circ t = t' \circ f_E$. Given a *graph of types* TG, a *typed graph* is a pair $\langle G, t_G\rangle$, where G is a graph and $t_G : G \to TG$ is a morphism. A morphism between typed graphs $f : \langle G_1, t_{G_1}\rangle \to \langle G_2, t_{G_2}\rangle$ is a graph morphisms $f : G_1 \to G_2$ consistent with the typing, i.e., such that $t_{G_1} = t_{G_2} \circ f$. The category of TG-typed graphs and typed graph morphisms is denoted by **TG-Graph**.

Fixed a graph TG of types, a *(TG-typed graph) production* $(L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ is a pair of injective typed graph morphisms $l : K \to L$ and $r : K \to R$. It is called *consuming* if morphism $l : K \to L$ is not surjective. The typed graphs L, K, and R are called the *left-hand side*, the *interface*, and the *right-hand side* of the production, respectively. A *(TG-typed) graph grammar* $\mathcal{G}$ is a tuple $\langle TG, G_{in}, P, \pi\rangle$, where G_{in} is the *initial (typed) graph*, P is a set of *production names*, and π a function which associates a graph production to each production name in P. We denote by $Elem(\mathcal{G})$ the set $N_{TG} \cup E_{TG} \cup P$. Moreover, sometimes we shall write $q : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ for $\pi(q) = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$.

Since in this paper we work only with typed notions, we will usually omit the qualification "typed", and we will not indicate explicitly the typing morphisms. Moreover, we will consider only *consuming* grammars, namely grammars where all productions are consuming: this corresponds, in the theory of Petri nets, to the common requirement that transitions must have non-empty preconditions.

Given a typed graph G, a production $q : (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$, and a *match* (i.e., a graph morphism) $g : L \to G$, a *direct derivation δ from G to H using q (based on g)* exists, written $\delta : G \Rightarrow_q H$, if and only if the diagram

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

can be constructed, where both squares have to be pushouts in **TG-Graph**.

Roughly speaking, the rewriting step removes from the graph G the items of the left-hand side which are not in the image of the interface, namely $L - l(K)$, producing in this way the graph D. Then the items in the right-hand side which are not in the image of the interface, namely $R - r(K)$, are added to D, obtaining the final graph H. Notice that the interface graph K (common part of L and R) specifies both what is preserved and how the added subgraph has to be connected to the remaining part.

It is worth recalling here that given an injective morphism $l : K \to L$ and a match $g : L \to G$ as in the above diagram, their *pushout complement* (i.e., a graph D with morphisms k and b such that the left square is a pushout) only exists if the *gluing conditions* are satisfied. These consist of two parts:

- the *identification condition*, requiring that if two distinct nodes or arcs of L are mapped by g to the same image, then both must be in the image of l;
- the *dangling condition*, stating that no arc in $G - g(L)$ should be incident to a node in $g(L - l(K))$ (because otherwise the application of the production would leave such an arc "dangling").

Notice that the identification condition does not forbid the match to be non-injective on preserved items. Intuitively this means that preserved (read-only) resources can be used with multiplicity greater than one.

A *derivation* over a grammar $\mathcal{G}$ is a sequence of direct derivations (over $\mathcal{G}$) $\rho = \{G_{i-1} \Rightarrow_{q_{i-1}} G_i\}_{i\in\{1,\ldots,n\}}$. The derivation is written as $\rho : G_0 \Rightarrow^*_{\{q_0,\ldots,q_{n-1}\}} G_n$ or simply as $\rho : G_0 \Rightarrow^* G_n$. 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.

Relation with Petri nets. To conclude this section it is worth explaining the relation between Petri nets and DPO graph grammars. The fact that graph transformation systems can model the behaviour of Petri nets has been first formalized by Kreowski in [8]. The proposed encoding of nets into grammars represents the topological structure of a marked net as a graph, in such a way that the firing of transitions is modelled by direct derivations.

Here we use a slightly simpler modelling, discussed, among others, in [2]. The basic observation is that a P/T Petri net is essentially a rewriting system on multisets, and that, given a set A, a multiset of A can be represented as a discrete graph typed over A. In this view a P/T Petri net can be seen as a graph grammar acting on discrete graphs typed over the set of places, the productions being (some encoding of) the net transitions: a marking is represented by a set of nodes (tokens) labelled by the place where they are, and, for example, the unique transition t of the net in Fig. 1.(a) is represented by the graph production in the top row of Fig. 1.(b). Notice that the interface is empty since nothing is explicitly preserved by a net transition.

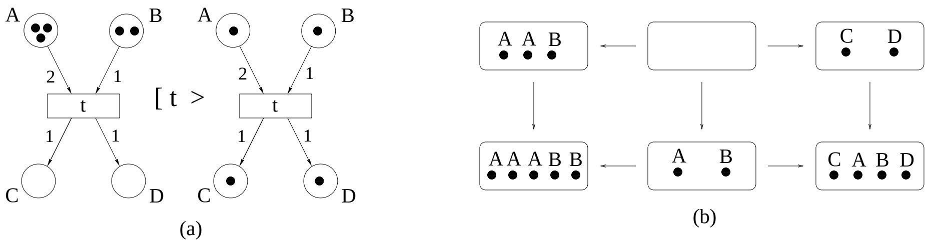

Fig. 1. Firing of a transition and corresponding DPO direct derivation.

It is easy to check that this representation satisfies the properties one would expect: a production can be applied to a given marking if and only if the corresponding transition is enabled and, in this case, the double pushout construction produces the same marking as the firing of the transition. For instance, the firing of transition t, leading from the marking $3A+2B$ to the marking $A+B+C+D$ in Fig. 1.(a), becomes the double pushout diagram of Fig. 1.(b).

The considered encoding of nets into grammars enlightens the dimensions in which graph grammars properly extend nets. First of all grammars allow for a more structured state, that is a general graph rather than a multiset (discrete graph). Perhaps more interestingly, graph grammars allow for productions where the interface graph may not be empty, thus specifying a "context" consisting of items that have to be present for the productions to be applied, but are not affected by the application. In this respect, graph grammars are closer to some generalizations of nets in the literature, called nets with read (test) arcs or contextual nets (see e.g. [7, 10, 14]), which generalize classical nets by adding the possibility of checking for the presence of tokens which are not consumed.

3 Nondeterministic occurrence grammars

The notion of derivation introduced in the previous section formalizes how a single computation of a grammar can evolve. Nondeterministic occurrence grammars are intended to represent the computations of graph grammars in a more static way, by recording the events (production applications) which can appear in all possible derivations and the dependency relations between them.

Analogously to what happens for nets, occurrence grammars are "safe" grammars, where the dependency relations between productions satisfy suitable acyclicity and well-foundedness requirements. However, while for nets it suffices to take into account only the causal dependency and the conflict relations, the greater complexity of grammars makes the situation much more involved. On the one hand, the fact that a production application not only consumes and produces, but also preserves a part of the state leads to a form of asymmetric conflict (or weak dependency) between productions. On the other hand, because of the dangling condition, also the graphical structure of the state imposes some precedences between productions.

A first step towards a definition of occurrence grammar is a suitable notion of safeness for grammars [4], generalizing the usual one for P/T nets, which requires that each place contains at most one token in any reachable marking.

Definition 1 ((strongly) safe grammar). *A grammar* $\mathcal{G} = \langle TG, G_{in}, P, \pi \rangle$ *is* (strongly) safe *if, for all* H *such that* $G_{in} \Rightarrow^* H$*,* H *has an injective typing morphism.*

Strongly safe graph grammars (hereinafter called just safe grammars) admit a natural net-like pictorial representation, where items of the type graph and productions play, respectively, the rôle of places and transitions of Petri nets. The basic observation is that typed graphs having an injective typing morphism can be safely identified with the corresponding subgraphs of the type graph (just thinking of injective morphisms as inclusions). Therefore, in particular, each graph $\langle G, t_G \rangle$ reachable in a safe grammar can be identified with the subgraph $t_G(G)$ of the type graph TG, and thus it can be represented by suitably decorating the nodes and arcs of TG. Concretely, a node is drawn as a filled circle if it belongs to $t_G(G)$ and as an empty circle otherwise, while an arc is drawn as a continuous line if it is in $t_G(G)$ and as a dotted line otherwise (see Fig. 2). This is analogous to the usual technique of representing the marking of a safe net by putting one token in each place which belongs to the marking.

With the above identification, in each computation of a safe grammar starting from the initial graph a production can only be applied to the subgraph of the type graph which is the image via the typing morphism of its left-hand side. Therefore according to its typing, we can safely think that a production *produces*, *preserves* or *consumes* items of the type graph. This is expressed by drawing productions as arrow-shaped boxes, connected to the consumed and produced resources by incoming and outcoming arrows, respectively, and to the preserved resources by undirected lines. Fig. 2 presents two examples of safe grammars, with their pictorial representation. Notice that the typing morphisms for the initial graph and the productions are represented by suitably labelling the involved graphs with items of the type graph.

Using a net-like language, we speak of *pre-set* ${}^\bullet q$, *context* $\underline{q}$ and *post-set* $q^\bullet$ of a production q, defined in the obvious way. Similarly for a node or arc x in TG we write ${}^\bullet x$, $\underline{x}$ and $x^\bullet$ to denote the sets of productions which produce, preserve and consume x. For instance, for grammar $\mathcal{G}_2$ in Fig. 2, the pre-set, context and post-set of production q_1 are ${}^\bullet q_1 = \{C\}$, $\underline{q_1} = \{B\}$ and ${q_1}^\bullet = \{A, L\}$, while for the node B, ${}^\bullet B = \emptyset$, $\underline{B} = \{q_1, q_2, q_3\}$ and $B^\bullet = \{q_4\}$.

Although the notion of causal relation is meaningful only for safe grammars, it is technically convenient to define it for general grammars. The same holds for the asymmetric conflict relation introduced below.

Definition 2 (causal relation). *The* causal relation *of a grammar $\mathcal{G}$ is the binary relation $<$ over $Elem(\mathcal{G})$ defined as the least transitive relation satisfying: for any node or arc x in the type graph TG, and for productions $q_1, q_2 \in P$*

1. *if $x \in {}^\bullet q_1$ then $x < q_1$;*
2. *if $x \in {q_1}^\bullet$ then $q_1 < x$;*
3. *if ${q_1}^\bullet \cap \underline{q_2} \neq \emptyset$ then $q_1 < q_2$;*

As usual $\leq$ is the reflexive closure of $<$. Moreover, for $x \in Elem(\mathcal{G})$ we denote by $\lfloor x \rfloor$ the set of causes of x in P, namely $\{q \in P : q \leq x\}$.

The first two clauses of the definition of relation $<$ 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, to be applied, requires that q_1 had already been applied.

Notice that the fact that an item is preserved by q_1 and consumed by q_2, i.e., $\underline{q_1} \cap {}^\bullet q_2 \neq \emptyset$ (e.g., the node B in grammar $\mathcal{G}_1$ of Fig. 2), does not imply $q_1 < q_2$. Actually, since q_1 must precede q_2 in any computation where both appear, in such computations q_1 acts as a cause of q_2. However, differently from a true cause, q_1 is not necessary for q_2 to be applied. Therefore we can think of

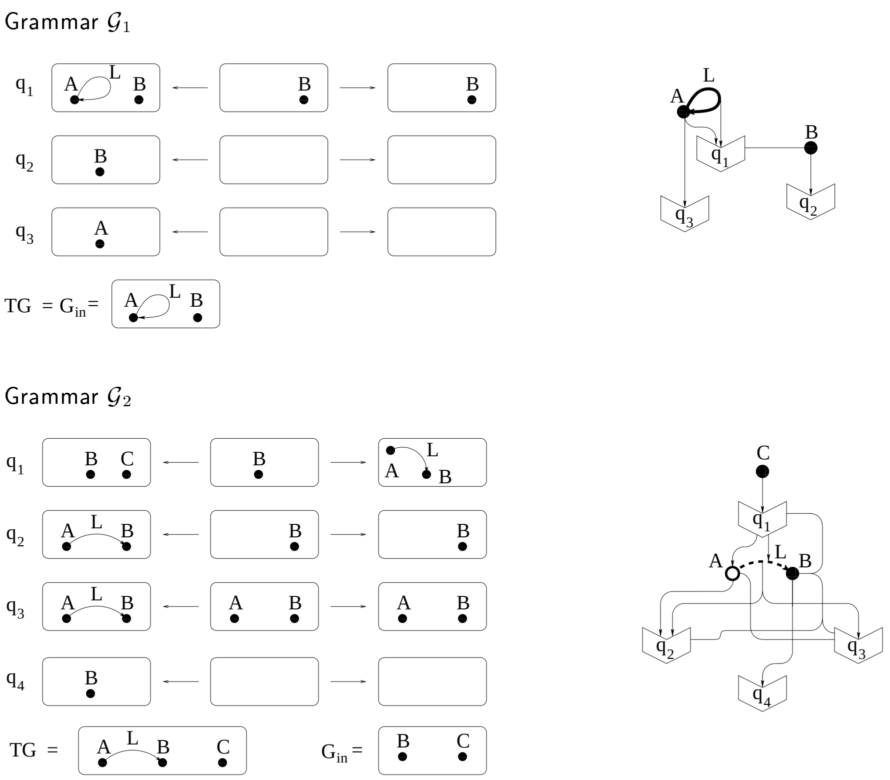

Fig. 2. Two safe grammars and their net-like representation.

the relation between the two productions as a *weak* form of *causal dependency*. Equivalently, we can observe that the application of q_2 prevents q_1 to be applied, so that q_1 can never follow q_2 in a derivation. But the converse is not true, since q_1 *can* be applied before q_2. Thus this situation can also be interpreted naturally as an *asymmetric conflict* between the two productions (see [1]).

Definition 3 (asymmetric conflict). *The* asymmetric conflict relation *of a grammar $\mathcal{G}$ is the binary relation $\nearrow$ over the set of productions, defined by:*

1. *if $\underline{q_1} \cap {}^\bullet q_2 \neq \emptyset$ then $q_1 \nearrow q_2$;*
2. *if ${}^\bullet q_1 \cap {}^\bullet q_2 \neq \emptyset$ and $q_1 \neq q_2$ then $q_1 \nearrow q_2$;*
3. *if $q_1 < q_2$ then $q_1 \nearrow q_2$.*

Condition 1 is justified by the discussion above. Condition 2 essentially expresses the fact that a situation of "classical" symmetric conflict is coded, in this setting, as an asymmetric conflict in both directions. Finally, since $<$ represents a global order of execution, while $\nearrow$ determines an order of execution only locally to each computation, it is natural to impose $\nearrow$ to be an extension of $<$ (Condition 3).

A *nondeterministic occurrence grammar* is an acyclic grammar which represents, in a branching structure, several possible computations starting from its initial graph and using each production at most once.

Definition 4 ((nondeterministic) occurrence grammar). *A* (nondeterministic) occurrence grammar *is a graph grammar* $\mathcal{O} = \langle TG, G_{in}, P, \pi \rangle$ *such that*

1. *its causal relation* $\leq$ *is a partial order, and for any* $q \in P$*, the set* $\lfloor q \rfloor$ *is finite and asymmetric conflict* $\nearrow$ *is acyclic on* $\lfloor q \rfloor$*;*
2. *the initial graph* G_{in} *coincides with the set* $Min(\mathcal{O})$ *of minimal elements of* $\langle Elem(\mathcal{O}), \leq \rangle$ *(with the graphical structure inherited from* TG *and typed by the inclusion);*
3. *each arc or node* x *in* TG *is created by at most one production in* P*, namely* $|\, {}^{\bullet}x \,| \leq 1$.
4. *for each production* $q : \langle L, t_L \rangle \stackrel{l}{\leftarrow} \langle K, t_K \rangle \stackrel{r}{\rightarrow} \langle R, t_R \rangle$*, the typing* t_L *is injective on the "consumed part"* $L - l(K)$*, and similarly* t_R *is injective on the "produced part"* $R - r(K)$.

Since the initial graph of an occurrence grammar $\mathcal{O}$ *is determined by* $Min(\mathcal{O})$*, we often do not mention it explicitly.*

One can show that, by the defining conditions, each occurrence grammar is *safe*.

Intuitively, conditions (1)–(3) recast in the framework of graph grammars the analogous conditions of occurrence nets (actually of occurrence contextual nets [1]). In particular condition (1) requires causality to be acyclic and each production q to have a finite set of causes $\lfloor q \rfloor$. Acyclicity of asymmetric conflict on $\lfloor q \rfloor$ corresponds to the requirement of irreflexivity for the conflict relation in occurrence nets. In fact, notice that if a set of productions forms an asymmetric conflict cycle $q_0 \nearrow q_1 \nearrow \ldots \nearrow q_n \nearrow q_0$, then such productions cannot appear in the same computation, otherwise the application of each production should precede the application of the production itself; this fact can be naturally interpreted as a form of n-ary conflict. Condition (2) forces the set of minimal items of the type graph to be a graph, coinciding with the initial graph of the grammar and Condition (3) requires the absence of backward conflicts. Condition (4), instead, is closely related to safeness and requires that each production consumes and produces items with multiplicity one. Together with acyclicity of $\nearrow$, it disallows the presence of some productions which surely could never be applied, because they fail to satisfy the identification condition with respect to the typing morphism.

On the contrary, the definition does not imply that every production of an occurrence grammar will ever satisfy the *dangling condition*. This fact deserves some comments since the dangling condition, which requires the *absence* of arcs pointing to nodes which are removed by the production, induces precedence relations on productions. For example, in the grammar $\mathcal{G}_2$ of Fig. 2 the application of production q_1 "disables" q_4, since q_4 would remove the node B, leaving the arc L dangling. The production q_4 becomes enabled again only after q_2 or q_3 has been applied. The reason why we defined occurrence grammars in this way is that the dangling condition is not purely syntactical and cannot be checked "locally" by looking only at the causes of the considered production. Checking such negative (non monotonic) condition on a production, would require to find a possible computation allowing for the execution of productions which remove the potentially dangling arcs, and to verify the consistency of such computation with the production at hand. It can be shown that such verification is, in general, exponential in the size of the occurrence grammar in the finite case. Even worse, for infinite occurrence grammars (which can be obtained as unfolding of finite grammars), the problem is undecidable, as it can be shown by using the Turing completeness of DPO graph grammars.

Disregarding the dangling condition has as a consequence the fact that, differently from what happens for occurrence nets, not every production in an occurrence grammar is guaranteed to be applicable at least in one derivation starting from the initial graph. The restrictions to the behaviour imposed by the dangling condition are considered when defining the configurations of an occurrence grammar, which represent exactly, in a sense formalized later, all the possible deterministic runs of the grammar.

Definition 5 (configuration). *A* configuration *of an occurrence grammar* $\mathcal{O} = \langle TG, P, \pi \rangle$ *is a subset* $C \subseteq P$ *such that*

1. *if* $\nearrow_C$ *denotes the restriction of the asymmetric conflict relation to* C*, then* $(\nearrow_C)^*$ *is a partial order, and* $\{q' \in C : q' (\nearrow_C)^* q\}$ *is finite for all* $q \in C$*;*[1]
2. C *is left-closed w.r.t.* $\leq$*, i.e. for all* $q \in C$*,* $q' \in P$*,* $q' \leq q$ *implies* $q' \in C$*;*
3. *for all* $e \in TG$ *and* $n \in \{s(e), t(e)\}$*, if* $n^\bullet \cap C \neq \emptyset$ *and* ${}^\bullet e \subseteq C$ *then* $e^\bullet \cap C \neq \emptyset$*.*

If C *satisfies conditions (1) and (2), then it is called a* pre-configuration*.*

The notion is reminiscent of that of configuration of asymmetric event structures and thus of occurrence contextual nets [1]. The first part of Condition 1 ensures that in C there are no $\nearrow$-cycles, and thus excludes the possibility of having in C a subset of productions in conflict. The second part guarantees that each production has to be preceded only by finitely many other productions in the computation represented by the configuration. Condition 2 requires the presence of all the causes of each production, while Condition 3 formalizes the dangling condition. If a configuration contains a production q consuming a node n and a production q' producing an arc e (i.e. ${}^\bullet e = \{q'\}$) with source (or target) n, then a production q'' removing such an arc must be present as well, otherwise, due to the dangling condition, q could not be executed. Notice that in this situation the production q'' can coincide with q itself; otherwise it surely preserves the node n and thus $q'' \nearrow q$, i.e. q'' correctly precedes q in the computation represented by the configuration. Similar considerations apply if the arc e is present in the initial graph, i.e., ${}^\bullet e = \emptyset$. For example the set of configurations of the grammar $\mathcal{G}_2$ in Fig. 2 is $\mathit{Conf}(\mathcal{G}_2) = \{\emptyset, \{q_1\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_2, q_4\}, \{q_1, q_3, q_4\}, \{q_4\}\}$. The set $S = \{q_1, q_4\}$, is instead only a pre-configuration, since for the node B we have $B = t(L)$, $q_4 \in B^\bullet$, ${}^\bullet L = \{q_1\} \subseteq S$, but the intersection of S with $L^\bullet = \{q_2, q_3\}$ is empty.

The fact that configurations represent all and only the deterministic runs of an occurrence grammar is formalized by the following result.

[1] As usual, for a binary relation r, with r^* we denote its transitive and reflexive closure.

Proposition 1 (configurations and derivations). *For any configuration C of an occurrence grammar $\mathcal{O}$, there exists a subgraph G_C of the type graph TG such that $Min(\mathcal{O}) \Rightarrow^*_C G_C$ with a derivation which applies exactly once every production in C, in any order consistent with $(\nearrow_C)^*$. Viceversa for each derivation $Min(\mathcal{O}) \Rightarrow^*_S G$ in $\mathcal{O}$, the set of productions S it applies is a configuration.*

As an immediate consequence of the previous result, a production which does not satisfy the dangling condition in any graph reachable from the initial graph is not part of any configuration. For example, q_3 does not appear in the set of configurations of $\mathcal{G}_1$, $\mathit{Conf}(\mathcal{G}_1) = \{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}\}$.

In the theory of Petri nets the notion of occurrence grammar is strictly related to that of process. A (non)deterministic net process is a (non)deterministic occurrence net with a morphism to the original net. Similarly, nondeterministic occurrence grammars can be used to define a suitable notion of *nondeterministic graph processes*, generalizing the deterministic graph processes of [4]. Then, the unfolding of a grammar, introduced in the next section, could be seen as a "complete" nondeterministic process of the grammar. Unfortunately, these notions cannot be discussed here because of space limitations.

4 Unfolding

This section introduces the unfolding construction which, applied to a consuming grammar $\mathcal{G}$, produces a nondeterministic occurrence grammar $\mathcal{U}_\mathcal{G}$ describing the behaviour of $\mathcal{G}$. The unfolding is equipped with a mapping $\phi_\mathcal{G}$ to the original grammar $\mathcal{G}$ which allows to see productions in $\mathcal{U}_\mathcal{G}$ as instances of production applications in $\mathcal{G}$, and items of the type graph of $\mathcal{U}_\mathcal{G}$ as instances of items of the type graph of $\mathcal{G}$.

The idea consists of starting from the initial graph of the grammar, then applying in all possible ways its productions, and recording in the unfolding each occurrence of production and each new graph item generated in the rewriting process, both enriched with the corresponding causal history. According to the discussion in the previous section, during the unfolding process productions are applied without considering the dangling condition. Moreover we adopt a notion of concurrency which is "approximated", again in the sense that it does not take care of the precedences between productions induced by the dangling condition.

Definition 6 (quasi-concurrent graph). *Let $\mathcal{O} = \langle TG, P, \pi \rangle$ be an occurrence grammar. A subgraph G of TG is called* quasi-concurrent *if*

1. $\bigcup_{x \in G} \lfloor x \rfloor$ *is a pre-configuration;*
2. $\neg(x < y)$ *for all* $x, y \in G$.

The intuitive idea is that each quasi-concurrent graph is contained in a graph reachable in a "lax version" of the DPO rewriting, where the dangling condition is not tested.

Another basic ingredient of the unfolding is the gluing operation. It can be interpreted as a "partial application" of a rule to a given match, in the sense that it generates the new items as specified by the production (i.e., items of right-hand side not in the interface), but items that should have been deleted are not affected: intuitively, this is because such items may still be used by another production in the nondeterministic unfolding. In the following we assume that for each production name q its associated production is $L_q \leftarrow K_q \rightarrow R_q$, where the injections l_q and r_q are inclusions (and not generic injective morphisms).

Definition 7 (gluing). *Let q be a production, G a graph and $m : L_q \rightarrow G$ a graph morphism. We define, for any symbol $*$, the* gluing of G and R_q along K_q, according to m and marked by $*$, *denoted by $glue_*(q, m, G)$ as the graph $\langle N, E, s, t\rangle$, where:*

$$N = N_G \cup m_*(N_{R_q}) \qquad\qquad E = E_G \cup m_*(E_{R_q})$$

with m_* defined by: $m_*(x) = m(x)$ if $x \in K_q$ and $m_*(x) = \langle x, *\rangle$ otherwise. The source and target functions s and t, and the typing are inherited from G and R_q.

The gluing operation keeps unchanged the identity of the items already in G, and records in each newly added item from R_q the given symbol $*$. We remark that the gluing, as just defined, is a concrete deterministic definition of the pushout of the arrows $G \overset{m}{\leftarrow} L_q \overset{l_q}{\hookleftarrow} K_q$ and $K_q \overset{r_q}{\hookrightarrow} R_q$.

Now the unfolding of a grammar $\mathcal{G} = \langle TG, G_{in}, P, \pi\rangle$ can be as follows. For each n, we construct a partial unfolding $\mathcal{U}(\mathcal{G})^{(n)} = \langle \mathcal{U}^{(n)}, \phi^{(n)}\rangle$, where $\mathcal{U}^{(n)} = \langle TG^{(n)}, P^{(n)}, \pi^{(n)}\rangle$ is an occurrence grammar and the mapping $\phi^{(n)} = \langle fp^{(n)}, fg^{(n)}\rangle$ consists of two components: a function $fp^{(n)} : P^{(n)} \rightarrow P$ mapping the productions of the unfolding into productions of $\mathcal{G}$, and a morphism $fg^{(n)} : TG^{(n)} \rightarrow TG$ from the type graph of $\mathcal{U}^{(n)}$ to TG. Intuitively, the occurrence grammar generated at level n contains all possible computations of the grammar with "causal depth" at most n.

- $(\mathbf{n = 0})$ $\langle TG^{(0)}, fg^{(0)}\rangle = G_{in}$, while $P^{(0)}$, $\pi^{(0)}$ and $fp^{(0)}$ are empty.
- $(\mathbf{n \rightarrow n + 1})$ Given $\mathcal{U}(\mathcal{G})^{(n)}$, the partial unfolding $\mathcal{U}(\mathcal{G})^{(n+1)}$ is obtained by extending it with all the possible production applications to quasi-concurrent subgraphs of the type graph of $\mathcal{U}^{(n)}$. More precisely, for each production $q \in P$ and match $m : L_q \rightarrow \langle TG^{(n)}, fg^{(n)}\rangle$ satisfying the identification condition, with $m(L_q)$ *quasi-concurrent* subgraph of $TG^{(n)}$:
  - Add to $P^{(n)}$ an occurrence of the production q, with name $q' = \langle q, m\rangle$. The match m is needed to record the "history" of q'. Now let $P^{(n)} := P^{(n)} \cup \{q'\}$, and extend $fp^{(n)}$ so that $fp^{(n)}(q') = q$. The production $\pi^{(n)}(q)$ coincides with $\pi(q)$ except for the typing.
  - Glue the type graph $TG^{(n)}$, typed over TG by $fg^{(n)}$, with the right-hand side R_q of q along K_q, according to the mapping m and marked by q'; in this way the new items generated by the production contain the name q' of the occurrence of the production and thus their history. The morphism $fg^{(n)}$ is updated consequently.

  After all the applicable productions have been considered we obtain $\mathcal{U}(\mathcal{G})^{(n+1)}$.

The deterministic gluing construction ensures that, at each step, the order in which productions are applied does not influence the final result of the step. Moreover if a production is applied twice (also in different steps) at the same match, the generated items are always the same and thus they appear only once in the unfolding.

Definition 8 (unfolding). *The unfolding $\mathcal{U}(\mathcal{G}) = \langle \mathcal{U}_{\mathcal{G}}, \phi_{\mathcal{G}} \rangle$ of the grammar $\mathcal{G}$ is defined as $\bigcup_n \mathcal{U}(\mathcal{G})^{(n)}$, where union is applied componentwise.*

It is not difficult to verify that for each n, $\mathcal{U}^{(n)}$ is a (finite) nondeterministic occurrence grammar, and $\mathcal{U}(\mathcal{G})^{(n)} \subseteq \mathcal{U}(\mathcal{G})^{(n+1)}$, componentwise. Therefore $\mathcal{U}_{\mathcal{G}}$ is an occurrence grammar. Moreover the unfolding process applied to an occurrence grammar yields a grammar which is isomorphic to the original one.

Finally, we notice that, as already remarked, not all productions in the unfolding are executable in some computation and thus correspond to occurrences of production of the original grammar. This is due to the fact that only the identification condition is tested and an "approximated notion" of concurrent subgraph is used. We stress that this is needed to have a decidable unfolding, a fact which, besides being pleasant from a purely theoretical point of view, is essential if one wants to use the unfolding in practice to prove properties of the modelled system.

5 Domain and event structure semantics

In the seminal work of Winskel on (safe) Petri nets, the unfolding semantics of a net, given in terms of a nondeterministic occurrence net, is further abstracted to an event structure semantics, by forgetting the "real structure" of the unfolding and recording only the relationships induced by such structure on the transitions of the unfolding itself. In this section we show that a similar construction can be carried out for graph grammars.

Recall that a *prime event structure with binary conflict (PES),* consists of a set of events endowed with two binary relations: a partial order relation $\leq$, modelling *causality*, and a symmetric and irreflexive relation $\#$, hereditary w.r.t. causality, modelling *conflict*. A *configuration* of a PES is a subset of events left-closed w.r.t. $\leq$ and conflict free, 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.

We already observed that the notion of configuration of an occurrence grammar allows us to recover exactly the different possible deterministic computations of the grammar. Following the ideas suggested for asymmetric event structures and contextual nets in [1], an order can be defined on configurations which captures the idea of computational extension. The main point is that, differently from what happens for classical event structures and Petri nets, due to the presence of the asymmetric conflict such an order is not simply set-inclusion: in fact, a configuration C cannot be extended with a production inhibited by some of the productions already present in C.

Definition 9 (poset of configurations). *Given an occurrence grammar $\mathcal{O}$, we denote by $Conf(\mathcal{O})$ the set of its configurations, ordered by the relation $\sqsubseteq$ defined as $C \sqsubseteq C'$ if $C \subseteq C'$ and $\neg(q' \nearrow q)$, for all $q \in C$ and $q' \in C' - C$.*

The partial order of configurations of an occurrence grammar exhibits a very nice algebraic structure, i.e., it is a domain. The proof (that we skip here) follows the same outline as in [1], but more effort is needed to take care of the additional requirement in the definition of configuration, related to the dangling condition.

Theorem 1 (from occurrence grammars to domains). *Given an occurrence grammar $\mathcal{O}$, the partial order of configurations $Conf(\mathcal{O})$ is a domain.*

By the relation between domains and event structures sketched above, $Conf(\mathcal{O})$ determines indirectly an event structure $ES(\mathcal{O})$, namely, the unique (up to isomorphisms) PES having $Conf(\mathcal{O})$ as domain of configurations. Differently from what happens for Petri nets, there is not a one to one correspondence between events of $ES(\mathcal{O})$ and productions in $\mathcal{O}$. Instead, a different event is generated for any possible "history" of each production of $\mathcal{O}$. This phenomenon of "duplication of events" is related to the fact that the new precedence relations arising between productions in graph grammars are represented via causality and conflict in classical PES's. Basically, a situation of asymmetric conflict like $q_1 \nearrow q_2$ in grammar $\mathcal{G}_1$ of Fig. 2, is coded in the PES by the insertion of a single event e_1 corresponding to q_1, and two "copies" e'_2 ad e''_2 of q_2, the first one in conflict with e_1 and the second one caused by e_1 (see Fig. 3.(a)). For what concerns the dangling condition, consider the grammar $\mathcal{G}_2$ in Fig. 2. In this case three conflicting events are generated corresponding to q_4: e_4 representing the execution of q_4 from the initial graph, which inhibits all other productions, and e'_4, e''_4 representing the execution of q_4 after q_2 and q_3, respectively.

Fig. 3. Coding asymmetric conflict and dangling condition in prime event structures.

As a final simple step, a domain and an event structure semantics for a graph grammar are readily defined via the unfolding construction.

Definition 10 (event structure semantics). *For any grammar $\mathcal{G}$, we denote by $Conf(\mathcal{G})$ the domain of configurations of the unfolding of $\mathcal{G}$, namely $Conf(\mathcal{U}_\mathcal{G})$, and by $ES(\mathcal{G})$ the corresponding event structure $ES(\mathcal{U}_\mathcal{G})$.*

6 Relation with other event structure semantics

This section briefly reviews two other event structure semantics proposed in the literature for DPO graph transformation systems. The first one [3] is built on top of the "abstract truly concurrent model of computation" of a grammar. The other one [13] is based on a deterministic variation of the DPO approach. Nicely, these two alternative event structures turn out to coincide with the one obtained from the unfolding, which thus can be claimed to give "the" event structure semantics of DPO graph transformation.

Event structure semantics from abstract derivations. The derivations of a grammar $\mathcal{G}$ are easily equipped with a simple algebraic structure which turns them into a category, called the *concrete model of computation for* $\mathcal{G}$ and denoted $\mathbf{Der}[\mathcal{G}]$. Objects in $\mathbf{Der}[\mathcal{G}]$ are graphs, and each derivation ρ is seen as an arrow from $\sigma(\rho)$ to $\tau(\rho)$. Given two derivations ρ and ρ' such that the ending graph of ρ and the starting graph of ρ' coincide, i.e., $\tau(\rho) = \sigma(\rho')$, their sequential composition $\rho\,;\,\rho'$ is the derivation obtained by identifying $\tau(\rho)$ with $\sigma(\rho')$.

The concrete model contains a lot of redundant information and it is far from representing what one has in mind as truly concurrent behaviour of the system modeled by the grammar. A more reasonable model, called the *abstract, truly concurrent model of computation* of a grammar $\mathcal{G}$, and denoted by $\mathbf{Tr}[\mathcal{G}]$, is the category obtained by imposing a suitable equivalence on objects and arrows of the concrete model. In particular, the objects of $\mathbf{Tr}[\mathcal{G}]$ are abstract graphs (i.e., isomorphism classes of graphs), while its arrows are *concatenable derivation traces*, i.e., equivalence classes of derivations with respect to the *concatenable truly concurrent equivalence* [3]. This equivalence is the least equivalence on derivations containing both the *abstraction equivalence*, a refinement of the obvious notion of derivation isomorphism compatible with sequential composition, and the *shift equivalence*, which equates two derivations if one can be obtained from the other by repeatedly shifting independent derivation steps.

The category $\mathbf{Tr}[\mathcal{G}]$ is used in [3] to define a domain and a prime event structure semantics for graph transformation systems. More precisely, for any consuming graph grammar $\mathcal{G} = \langle TG, G_{in}, P, \pi\rangle$, one considers the comma category $([G_{in}] \downarrow \mathbf{Tr}[\mathcal{G}])$, where objects are concatenable derivation traces of $\mathbf{Tr}[\mathcal{G}]$ with source in $[G_{in}]$, and given two such traces δ_0 and δ_1, an arrow from δ_0 to δ_1 is a concatenable derivation trace δ satisfying $\delta_0;\delta = \delta_1$. Such category can be shown to be 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 $\mathbf{Dom}[\mathcal{G}]$ and proposed as truly concurrent semantics of the grammar.

As announced such domain semantics can be proved to coincide with the one obtained from the unfolding $\mathcal{U}(\mathcal{G})$. In fact, we know from [3] that the finite elements of the domain $\mathbf{Dom}[\mathcal{G}]$ are one-to-one with derivation traces of $\mathcal{G}$ having the initial graph as source. Then the result is proved by showing that a bijection can be defined between finite elements of $Conf(\mathcal{G})$ and such derivation traces. In one direction, given a finite configuration $C \in Conf(\mathcal{G})$, consider any derivation $Min(\mathcal{U}_\mathcal{G}) \Rightarrow^*_C G_C$ in $\mathcal{U}_\mathcal{G}$ which applies exactly once every production in C, in any order consistent with the asymmetric conflict $\nearrow$. Such derivation, typed over the type graph of $\mathcal{G}$ via the mapping $\phi_\mathcal{G}$, gives a derivation d in $\mathcal{G}$, which determines the derivation trace associated to C. Viceversa, given a derivation trace $[d]$ of $\mathcal{G}$, the corresponding configuration of $\mathcal{U}_\mathcal{G}$ is determined as the set of productions needed to "simulate" d in the unfolding of $\mathcal{G}$. The fact that the ordering on configurations is not simply set-inclusion then plays a key rôle in the proof that such bijection is an isomorphism of partial orders.

Theorem 2. *For any (consuming) graph grammar $\mathcal{G}$, the domains $Conf(\mathcal{G})$ and $\mathbf{Dom}[\mathcal{G}]$ are isomorphic.*

Event structure semantics from deterministic derivations. Schied in [13] proposes a construction for defining an event structure semantics for *distributed rewriting systems*, an abstract unified model where several kind of rewriting systems, such as graph grammars and term rewriting systems, naturally fit. He shows that, given a distributed rewriting system $\mathcal{R}$, a domain $\mathcal{T}_\mathcal{R}$ can be obtained as the quotient, with respect to shift equivalence, of the collection of derivations starting from the initial state, ordered by the prefix relation. To prove the algebraic properties of $\mathcal{T}_\mathcal{R}$ he constructs, as an intermediate step, a *trace language* based on the shift equivalence, and applies general results to extract an event structure $\mathcal{E}_\mathcal{R}$ from the trace language. Finally he shows that $\mathcal{T}_\mathcal{R}$ is isomorphic to the domain of configurations of $\mathcal{E}_\mathcal{R}$.

The main interest in Schied's paper is for the application to graph grammars. Let us sketch how, according to Schied, the above construction instantiates to the case of grammars. Graph grammars are modeled as distributed rewriting systems by considering a *deterministic* variation of the DPO approach, where at each direct derivation the derived graph is uniquely determined by the host graph, the applied production and the match. The idea consists of working on concrete graphs, where each item records his causal history. Formally the definition of *deterministic direct derivation* (adapted to the typed case) is as follows.

Definition 11 (deterministic derivation). *Let $q : L_q \leftarrow K_q \rightarrow R_q$ be a production and let $m : L_q \rightarrow G$ be a match. Then a* deterministic direct derivation *$G \rightsquigarrow_{q,m} H$ exists if m satisfies the gluing conditions and*

$$H = glue_{\langle q,m \rangle}(q, m, G) - m(L_q - l(K_q)).$$

Let $\mathcal{G} = \langle TG, G_{in}, P, \pi \rangle$ be a typed graph grammar. A deterministic derivation *in $\mathcal{G}$ is a sequence of deterministic direct derivations $G_{in} \rightsquigarrow_{q_1,m_1} G_1 \rightsquigarrow_{q_2,m_2} \ldots \rightsquigarrow_{q_n,m_n} G_n$, starting from the initial graph and applying productions of $\mathcal{G}$.*

The construction of the domain of a grammar is based on the partial order of deterministic derivations with the prefix relation, and on shift equivalence.

Definition 12 (Schied's domain). *The Schied's domain for a consuming grammar $\mathcal{G}$, denoted by $\mathcal{T}_{\mathcal{G}}$, is defined as the quotient, w.r.t. shift equivalence, of the partial order of deterministic derivations of a grammar $\mathcal{G}$.*

It is not difficult to see that the (ideal completion of) Schied's domain for a grammar coincides with the domain of configurations of its unfolding $Conf(\mathcal{G})$, and thus with the domain $\mathbf{Dom}[\mathcal{G}]$ of [3]. The bijection between $\mathcal{T}_{\mathcal{G}}$ and the finite elements of $Conf(\mathcal{G})$ associates to the class of shift equivalent deterministic derivations containing $d : G_{in} \leadsto_{q_1,m_1} G_1 \leadsto_{q_2,m_2} \ldots \leadsto_{q_n,m_n} G_n$ the set $\{\langle q_i, m_i \rangle : i \in \underline{n}\}$, which can be shown to be a configuration in the unfolding of $\mathcal{G}$, independently of the particular derivation picked up in the class.

Theorem 3. *For any graph grammar $\mathcal{G}$, the ideal completion of $\mathcal{T}_{\mathcal{G}}$ and the domain $Conf(\mathcal{G})$ are isomorphic.*

7 Conclusions and future work

This paper introduces a notion of *nondeterministic* occurrence grammar for graph transformation systems in the algebraic DPO approach, by extending the work developed in [4] for the deterministic case. The phenomenon of asymmetric conflict between productions, caused by the possibility of performing "context sensitive" rewritings, cannot be ignored in this nondeterministic setting, and comes into play as an essential ingredient. A new kind of dependency between productions is also induced by the dangling condition, which imposes precedences among productions finalized at preserving the consistency of the graphical structure of the state.

Following the classical idea proposed by Winskel [15] for Petri nets, an *unfolding semantics* for DPO graph rewriting systems has been defined as a nondeterministic occurrence grammar, representing, in a single "branching" structure, all the possible computations of the grammar. The dangling condition, being a negative (non monotone) condition, can hardly be verified during the unfolding process. As a consequence the generated unfolding contains some garbage of which we get rid only when considering the set of configurations.

Interestingly, the set of configurations $Conf(\mathcal{G})$ of (the unfolding of) a grammar $\mathcal{G}$, suitably ordered using the asymmetric conflict relation, turns out to be a (finitary pairwise coherent) prime algebraic domain, one of the most widely used mathematical structures in the semantics of concurrency, equivalent to prime event structures (with binary conflict). Such domain is shown to coincide both with the domain $\mathbf{Dom}[\mathcal{G}]$, built from the category of concatenable derivation traces and proposed as semantics of a grammar in [3], and with the domain defined by Schied [13] and based on a concrete formulation of the DPO rewriting.

Finally, it is worth mentioning that the original work of Winskel shows that the unfolding construction extends to a coreflection from the category of safe nets to the category of domains, while our construction has been defined, up to now, only at "object level". We are working to obtain a full correspondence with Winskel's construction for nets, by extending the results presented in this paper to a categorical "in the large" level. Some suggestions can surely come from [1], where Winskel's construction has already been fully extended to contextual nets.

Acknowledgements

The authors wish to thank Roberto Bruni and Fabio Gadducci for helpful discussions and the anonymous referees for their comments on the submitted version of this paper.

References

1. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. *Proceedings of FoSSaCS '98*, volume 1378, pages 63–80. Springer, 1998.
2. A. Corradini. Concurrent Graph and Term Graph Rewriting. *Proceedings CONCUR'96*, volume 1119 of *LNCS*, pages 438–464. Springer, 1996.
3. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An Event Structure Semantics for Graph Grammars with Parallel Productions. *Proceedings of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *LNCS*. Springer, 1996.
4. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
5. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations.* World Scientific, 1997.
6. H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 3–14. Springer, 1987.
7. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.
8. H.-J. Kreowski. A comparison between Petri nets and graph grammars. *Proceedings of the Workshop on Graphtheoretic Concepts in Computer Science*, volume 100 of *LNCS*, pages 306–317. Springer, 1981.
9. 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.
10. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32, 1995.
11. W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer, 1985.
12. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* PhD thesis, Technische Universität Berlin, 1996.
13. G. Schied. On relating Rewriting Systems and Graph Grammars to Event Structures. *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 326–340. Springer, 1994.
14. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. Technical Report 352, Institüt für Mathematik, Augsburg University, 1996.
15. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.