Coreflective Concurrent Semantics for Single-Pushout Graph Grammars*

Paolo Baldan[1], Andrea Corradini[2], Ugo Montanari[2], Leila Ribeiro[3]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italia.
[2] Dipartimento di Informatica, Università di Pisa, Italia.
[3] Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil
baldan@dsi.unive.it, { andrea, ugo }@di.unipi.it, leila@inf.ufrgs.br

Abstract. The problem of extending to graph grammars the unfolding semantics originally developed by Winskel for (safe) Petri nets has been faced several times along the years, both for the single-pushout and double-pushout approaches, but only partial results were obtained. In this paper we fully extend Winskel's approach to single-pushout grammars providing them with a categorical concurrent semantics expressed as a coreflection between the category of graph grammars and the category of prime algebraic domains.

Introduction

It belongs to the folklore that Graph Grammars [25] generalise Petri nets, in that they allow for a more structured representation of system states, modelled in terms of graphs rather than (multi)sets, and for a more general kind of state transformation, modelling also preservation of parts of the state, besides deletion and creation.

During the last years, a rich theory of concurrency for the *algebraic approaches* to graph transformation has been developed, including the generalisation of various classical Petri net concurrency models, like Goltz-Reisig process semantics [13] and Winskel's unfolding semantics [27].

Recall that, building on [22], the seminal work [27] gives the concurrent semantics of (safe) nets by means of a chain of coreflections leading from the category of safe Petri nets to the category of prime algebraic domains.

$$\textbf{Safe Nets} \underset{\mathcal{U}}{\overset{\perp}{\leftrightarrows}} \textbf{Occurrence Nets} \underset{\mathcal{E}}{\overset{\mathcal{N}}{\leftrightarrows}} \textbf{Prime Event Structures} \underset{\mathcal{L}}{\overset{\mathcal{P}}{\leftrightarrows}} \textbf{Domains}$$

The first step unfolds any (safe) net into an occurrence net, i.e., a branching acyclic net making explicit causality and conflict (nondeterministic choice point) between events in the net. The second step produces a *prime event structure*

* Research supported by the FET-GC Project IST-2001-32747 AGILE and by the MIUR Project COFIN 2001013518 COMETA.

(PES) abstracting away the state and recording only the events and the relationships between events. Finally, the last step maps any PES into the corresponding prime algebraic domain of configurations.

Some important steps have been taken in the direction of developing an analogous semantical framework for algebraic graph grammars, but a definitive answer has not been provided yet. More precisely, a number of constructions have been defined for algebraic, *double-pushout* (DPO) graph grammars [12, 9] by the first three authors (see, e.g., [1]), as summarised by the following diagram:

$$\textbf{DPO Graph Grammars} \underset{\mathcal{U}_g}{\overset{\perp}{\hookleftarrow}} \textbf{Occurrence Grammars} \xrightarrow[\mathcal{E}_g]{} \textbf{Inhibitor Event Structures} \underset{\mathcal{L}_i}{\overset{\mathcal{P}_i}{\leftarrow\!\!\perp\!\!\rightarrow}} \textbf{Domains}$$

Even if at this level of abstraction it is not possible to see the relevant differences in the technical treatment of DPO grammars w.r.t. the much simpler case of Petri nets, still it is worth pointing at the evident differences between this chain of categories and the corresponding one for nets. Firstly, the category of PES's is replaced by that of *inhibitor event structures* (IES's), which, assuming conditional or-causality as a basic relation between events, are able to capture both the asymmetric conflicts between events arising from the capability of preserving part of the state and the inhibiting effects related to the presence of the application conditions for rules. The category of domains can be viewed as a coreflective subcategory of IES's (as shown by the last step of the chain) and thus one can also recover a semantics for DPO grammars in terms of domains and PES's. Secondly, the functor from the category of occurrence grammars to the category of IES's *does not admit* a left adjoint establishing a coreflection between IES's and occurrence grammars, and thus the whole semantic transformation is not expressed as a coreflection.

In this paper we concentrate on the *single-pushout* (SPO) approach [18, 11] to graph transformation. One of the main differences with respect to the DPO approach lies in the fact that there are no conditions on rule application, i.e., whenever a match is found the corresponding rule can always be applied. For SPO grammars an unfolding construction has been proposed in [24], corresponding to the first step in the above chains of coreflections.

Building on the results briefly summarised above, we provide a coreflective unfolding semantics for SPO graph grammars, defined through the following chain of coreflections:

$$\textbf{SPO Graph Grammars} \underset{\mathcal{U}_s}{\overset{\perp}{\hookleftarrow}} \textbf{Occurrence Grammars} \underset{\mathcal{E}_s}{\overset{\mathcal{N}_s}{\leftarrow\!\!\perp\!\!\rightarrow}} \textbf{Asymmetric Event Structures} \underset{\mathcal{L}_a}{\overset{\mathcal{P}_a}{\leftarrow\!\!\perp\!\!\rightarrow}} \textbf{Domains}$$

In particular, this construction differs from and improves that for DPO graph grammars, discussed above, because of the following facts:

– Due to the absence of application conditions for rules, a less powerful and more manageable kind of event structures called *asymmetric event structures* (introduced to deal with contextual nets in [4]), can be used to represent the dependency structure of SPO graph grammars.

– A novel construction, inspired by the work on contextual nets, allows to associate a canonical occurrence SPO graph grammar to any asymmetric event structure. This provides the lacking step, i.e., a left adjoint functor establishing a coreflection between the category of occurrence graph grammars and the category of asymmetric event structures.

An existing result [4] establishes a coreflection between asymmetric event structures and domains, so that we obtain a coreflective PES and domain semantics for SPO graph grammars.

These results do not extend immediately to the DPO approach because of the presence of application conditions for rules. However, as discussed in the conclusions, they can give some suggestions for improving the treatment of this more complex case.

The rest of the paper is structured as follows. In Section 1 we review the basics of single-pushout graph grammars and we define the notion of graph grammar morphism we shall work with. In Section 2 we discuss the kind of dependencies arising between events in SPO graph grammars and we introduce the notion of occurrence graph grammar. In Section 3 we briefly discuss the unfolding construction for SPO graph grammars and its characterisation as a universal construction. In Section 4 we complete the chain of coreflections from grammars to domains, showing how any occurrence grammar can be abstracted to an asymmetric event structure and, vice versa, how a canonical occurrence grammar can be associated to any asymmetric event structure. Finally Section 5 draws some conclusions.

1 Typed graph grammars and their morphisms

In this section we summarise the basics of graph grammars in the *single-pushout* (SPO) approach [18], an algebraic approach to graph rewriting alternative to the classical *double-pushout* (DPO) approach. The original SPO approach is adapted to deal with *typed graphs* [8, 19], which are, roughly, graphs labelled over a structure (the *graph of types*) that is itself a graph. Then some insights are provided on the relationship between typed graph grammars and Petri nets. Finally, the class of SPO typed graph grammars is turned into a category **GG** by defining a notion of grammar morphism, which recasts in this setting the morphisms for DPO grammars introduced in [3].

1.1 Typed graph grammars

Given a partial function $f : A \rightarrowtail B$ we will denote by $dom(f)$ its *domain*, i.e., the set $\{a \in A \mid f(a) \text{ is defined}\}$. Let $f, g : A \rightarrowtail B$ be two partial functions. We will write $f \leq g$ when $dom(f) \subseteq dom(g)$ and $f(x) = g(x)$ for all $x \in dom(f)$.

For a graph G we will denote by N_G and E_G the sets of *nodes* and *edges* of G, and by $s_G, t_G : E_G \to N_G$ its *source* and *target* functions.

Definition 1 (partial graph morphism). *A* partial graph morphism $f : G \rightarrowtail H$ *is a pair of partial functions* $f = \langle f_N : N_G \rightarrowtail N_H, f_E : E_G \rightarrowtail E_H \rangle$ *such that (see Fig. 1.(a)):*

$$s_H \circ f_E \leq f_N \circ s_G \quad \text{and} \quad t_H \circ f_E \leq f_N \circ t_G. \ (*)$$

We denote by **PGraph** *the category of (directed, unlabelled) graphs and partial graph morphisms. A morphism is called* total *if both components are total, and the corresponding full subcategory of* **PGraph** *is denoted by* **Graph**.

Notice that, according to condition (*), if f is defined over an edge then it must be defined both on its source and target nodes: this ensures that the domain of f is a well-formed graph. The inequalities in condition (*) ensure that *any* subgraph of a graph G can be the domain of a partial morphism $f : G \rightarrowtail H$. Instead, the stronger (apparently natural) conditions $s_H \circ f_E = f_N \circ s_G$ and $t_H \circ f_E = f_N \circ t_G$ would have imposed f to be defined over an edge whenever it is defined either on its source or on its target node.

Given a graph TG, a *typed graph* G over TG is a graph $|G|$, together with a total morphism $t_G : |G| \rightarrow TG$. A *partial morphism* between TG-typed graphs $f : G_1 \rightarrowtail G_2$ is a partial graph morphisms $f : |G_1| \rightarrowtail |G_2|$ consistent with the typing, i.e., such that $t_{G_1} \geq t_{G_2} \circ f$ (see Fig. 1.(b)). A typed graph G is called *injective* if the typing morphism t_G is injective. The category of TG-typed graphs and partial typed graph morphisms is denoted by TG-**PGraph**.

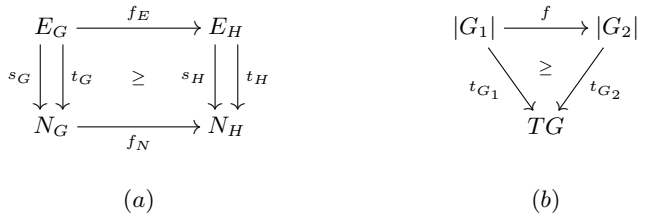


Fig. 1. Diagrams for partial graph and typed graph morphisms.

Given a partial typed graph morphism $f : G_1 \rightarrowtail G_2$, we denote by $dom(f)$ the domain of f typed in the obvious way.

Definition 2 (graph production and direct derivation). *Fixed a graph TG of types, a* (TG-typed graph) production q *is an injective partial typed graph morphism* $L_q \overset{r_q}{\rightarrowtail} R_q$. *It is called* consuming *if the morphism is not total. The typed graphs L_q and R_q are called the* left-hand side *and the* right-hand side *of the production, respectively.*

Given a typed graph G and a match *(i.e., a total morphism) $g : L_q \rightarrow G$, we say that there is a* direct derivation δ from G to H using q (based on g)*, written*

$\delta : G \Rightarrow_q H$, *if the following is a pushout square in* TG-**PGraph**.

$$\begin{array}{ccc} L_q & \xrightarrow{r_q} & R_q \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ G & \xrightarrow[d]{} & H \end{array}$$

Roughly speaking, the rewriting step removes from the graph G the image of the items of the left-hand side which are not in the domain of r_q, namely $g(L_q - dom(r_q))$, adding the items of the right-hand side which are not in the image of r_q, namely $R_q - dom(r_q)$. The items in the image of $dom(r_q)$ are "preserved" by the rewriting step (intuitively, they are accessed in a "read-only" manner).

A relevant difference with respect to the DPO approach is that here there is no *dangling condition* [9] preventing a rule to be applied whenever its application would leave dangling edges. In fact, as a consequence of the way pushouts are constructed in TG-**PGraph**, when a node is deleted by the application of a rule also all the edges having such node as source or target are deleted by the rewriting step, as a kind of *side-effect*. For instance, production q in the top row of Fig. 2, which consumes node B, can be applied to the graph G in the same figure. As a result both node B and the loop edge L are removed.

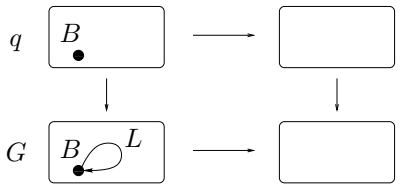


Fig. 2. Side-effects in SPO rewriting.

Even if the category **PGraph** has all pushouts, still we will consider a condition which corresponds to the *identification condition* of the DPO approach.

Definition 3 (valid match). *A match* $g : L_q \to G$ *is called* valid *when for any* $x, y \in |L_q|$, *if* $g(x) = g(y)$ *then* $x, y \in dom(r_q)$.

Conceptually, a match is not valid if it requires a single resource to be consumed twice, or to be consumed and preserved at the same time.

Definition 4 (typed graph grammar and derivation). *A* (TG-typed) SPO graph grammar $\mathcal{G}$ *is a tuple* $\langle TG, G_s, P, \pi \rangle$, *where* G_s *is the* (typed) start graph, P *is a set of* production names, *and* π *is a function which associates a production to each name in* P. *A graph grammar is* consuming *if all the productions in the range of* π *are consuming. A* derivation *in* $\mathcal{G}$ *is a sequence of direct derivations beginning from the start graph* $\rho = \{G_{i-1} \Rightarrow_{q_{i-1}} G_i\}_{i \in \{1,\ldots,n\}}$, *with* $G_0 = G_s$. *A derivation is* valid *if so are all the matches in its direct derivations.*

In the paper we will consider only *consuming* graph grammars and *valid* derivations. The restriction to consuming grammars is essential to obtain a meaningful semantics combining concurrency and nondeterminism. In fact, the presence of non-consuming productions, which can be applied without deleting any item, would lead to an unbounded number of concurrent events with the same causal history. This would not fit with the approach to concurrency (see, e.g., [13, 27]) where events in computations are identified with their causal history (formally, the unfolding construction would not work). On the other hand, considering valid derivations only, is needed to have a computational interpretation which is resource-conscious, i.e., where a resource can be consumed only once.

We denote by $Elem(\mathcal{G})$ the set $N_{TG} \cup E_{TG} \cup P$. We will assume that for each production name q the corresponding production $\pi(q)$ is $L_q \overset{r_q}{\rightarrowtail} R_q$. Without loss of generality, we will assume that the injective partial morphism r_q is a partial inclusion (i.e., that $r_q(x) = x$ whenever defined).

1.2 Relation with Petri nets

The reader who is familiar with Petri net theory can gain a solid intuition about grammar morphisms and many other definitions and constructions in this paper, by referring to the relation between Petri nets and (SPO) graph grammars. The correspondence between these two formalisms (see, e.g., [6] and references therein) relies on the basic observation that a P/T Petri net is essentially a rewriting system on a restricted kind of graphs, namely discrete, labelled graphs (that can be identified with sets of tokens labelled by places), the productions being the net transitions.

For instance, Fig. 3 presents a Petri net transition t and the corresponding graph production r_t which consumes nodes corresponding to two tokens in s_0 and one token in s_1 and produces new nodes corresponding to one token in s_2 and one token in s_3. The domain of the rule morphism is empty, i.e., $r_t : L \rightarrowtail R$ is the empty function, since nothing is explicitly preserved by a net transition.

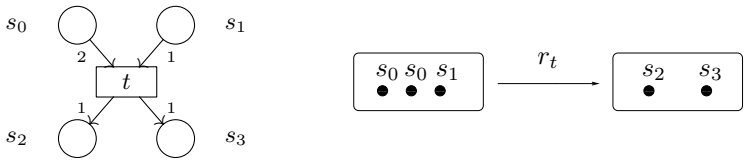


Fig. 3. A Petri net transition and a corresponding SPO production.

Note that, in this encoding of transitions into productions, the restriction to consuming graph grammars corresponds, in the theory of Petri nets, to the common requirement that transitions must have non-empty preconditions.

A tighter correspondence can be established with *contextual nets* [21], also called nets with test arcs in [5], activator arcs in [15] or read arcs in [26], an extension ordinary nets with the possibility of checking for the presence of tokens which are not consumed. Non-directed (usually horizontal) arcs are used to represent context conditions. For instance, transition t in the left part of Fig. 4 has place s as context, hence at least one token in s is needed for enabling t, and the firing of t does not affect such token.

As shown in Fig. 4, the context of a transition t in a contextual net corresponds to the graph $dom(r_t)$ of the corresponding SPO production $r_t : L \rightarrowtail R$. Thus, in general, a contextual net corresponds to an SPO graph grammar still acting on discrete graphs, but where a production may preserve some nodes, i.e., its domain might not be empty.

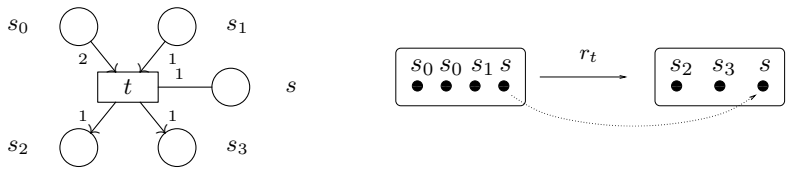


Fig. 4. A contextual Petri net transition and a corresponding SPO production.

1.3 Grammar morphisms

The notion of SPO grammar morphism defined in this paper recasts in the setting of the SPO approach the notion introduced for DPO grammars in [7, 3], which in turn was a generalisation of Petri net morphisms. Recall that a Petri net morphism [27] consists of two components: a multirelation between the sets of places, and a partial function mapping transitions of the first net into transitions of the second one. Net morphisms are required to "preserve" the algebraic structure of a net in the sense that the pre- (post-)set of the image of a transition t must be the image of the pre- (post-)set of t.

Recall that, given two sets A and B, a *multirelation* $R : A \leftrightarrow B$ is a function $R : A \times B \to \mathbb{N}$. Intuitively, R relates elements $a \in A$ and $b \in B$ with multiplicity $R(a, b)$. As the items of the type graph of a graph grammar can be seen as generalisations of Petri net places and typed graphs as generalisations of multisets of places, the first component of a grammar morphism will be a span of total graph morphisms between the type graphs of the source and target grammars, arising as a categorical generalisation of the notion of multirelation. Here we give only some basic definitions. For an extensive discussion we refer the reader to [7, 1].

Definition 5 (spans). *Let* $\mathbf{C}$ *be a category. A* (concrete) span *in* $\mathbf{C}$ *is a pair of coinitial arrows* $f = \langle f^L, f^R \rangle$ *with* $f^L : x_f \to a$ *and* $f^R : x_f \to b$. *Objects* a

and b are called the source and the target of the span, written $f : a \leftrightarrow b$. The span f will be sometimes denoted as $\langle f^L, x_f, f^R \rangle$, explicitly giving the common source object x_f.

Consider the equivalence $\sim$ over the set of spans with the same source and target defined, for $f, f' : a \leftrightarrow b$, as $f \sim f'$ if there exists an isomorphism $k : x_f \to x_{f'}$ such that $f'^L \circ k = f^L$ and $f'^R \circ k = f^R$ (see Fig. 6.(a)). The isomorphism class of a span f is denoted by $[f]$ and called a semi-abstract span.

Fig. 5 gives two examples of multirelations in **Set**, with the corresponding span representations.

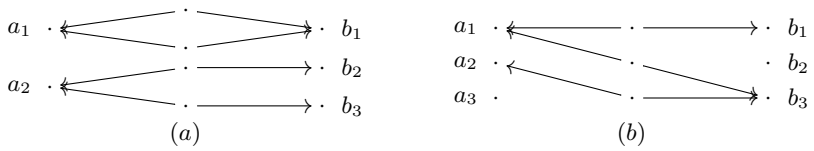


Fig. 5. The (semi-abstract) spans for the multirelations (a) $R_1(a_1, b_1) = 2$, $R_1(a_2, b_2) = 1$, $R_1(a_2, b_3) = 1$ and (b) $R_2(a_1, b_1) = 1$, $R_2(a_1, b_3) = 1$, $R_2(a_2, b_3) = 1$ (Pairs which are not mentioned are mapped to 0).

Definition 6 (category of spans). *Let $\mathbf{C}$ be a category with pullbacks. Then the category* $\mathbf{Span}(\mathbf{C})$ *has the same objects of $\mathbf{C}$ and semi-abstract spans on $\mathbf{C}$ as arrows. More precisely, a semi-abstract span $[f]$ is an arrow from the source to the target of f. The composition of two semi-abstract spans $[f_1] : a \leftrightarrow b$ and $[f_2] : b \leftrightarrow c$ is the (equivalence class) of a span f constructed as in Fig. 6.(b) (i.e., $f^L = f_1^L \circ y$ and $f^R = f_2^R \circ z$), where the square is a pullback. The identity on an object a is the equivalence class of the span $\langle id_a, id_a \rangle$, where id_a is the identity of a in $\mathbf{C}$.*

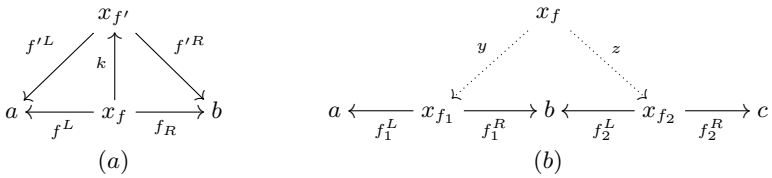


Fig. 6. Equivalence and composition of spans.

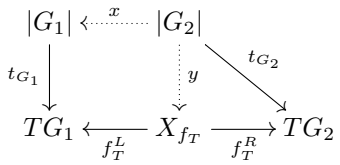
Relations can be identified with special multirelations $R : A \leftrightarrow B$ where multiplicities are bounded by one (namely $R(a, b) \leq 1$ for all $a \in A$ and $b \in B$).

The corresponding condition on a span $f : A \leftrightarrow B$ is the existence of at most one path between any two elements $a \in A$ and $b \in B$. For instance, the span in Fig. 5.(a) is not relational, while that in Fig. 5.(b) is relational.

Definition 7 (relational span). *Let* $\mathbf{C}$ *be a category. A span* $f : a \leftrightarrow b$ *in* $\mathbf{C}$ *is called* relational *if* $\langle f^L, f^R \rangle : x_f \to a \times b$ *is mono.*

We can also find a categorical analogue of constructing the image of a multiset through a multirelation. The next definition is given for graphs, but it could be generalised to any category with pullbacks.

Definition 8 (pullback-retyping construction). *Let* $[f_T] : TG_1 \leftrightarrow TG_2$ *be a semi-abstract span in* **Graph** *and let* G_1 *be a* TG_1*-typed graph. Then* G_1 *can be "transformed" into a* TG_2*-typed graph as depicted in the diagram below, by first taking a pullback (in* **Graph***) of the arrows* $f_T^L : X_{f_T} \to TG_1$ *and* $t_{G_1} : |G_1| \to TG_1$*, and then typing the pullback object over* TG_2 *by using the right part of the span* $f_T^R : X_{f_T} \to TG_2$*.*



The TG_2*-typed graph* $G_2 = \langle |G_2|, f_T^R \circ y \rangle$ *is determined only up to isomorphism. Sometimes we will write* $f_T\{x, y\}(G_1, G_2)$ *(or simply* $f_T(G_1, G_2)$ *if we are not interested in morphisms* x *and* y*) to express the fact that* G_1 *and* G_2 *are related in this way by the pullback-retyping construction induced by* $[f_T]$*.*

We are now ready to define grammar morphisms. Besides the component specifying the relation between the type graphs, a morphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ includes a (partial) mapping between production names. Furthermore a third component explicitly relates the (untyped) graphs underlying corresponding productions of the two grammars, as well as the graphs underlying the start graphs.

Definition 9 (grammar morphism). *Let* $\mathcal{G}_i = \langle TG_i, G_{s_i}, P_i, \pi_i \rangle$ *(*$i \in \{1, 2\}$*) be two graph grammars. A morphism* $f : \mathcal{G}_1 \to \mathcal{G}_2$ *is a triple* $\langle [f_T], f_P, \iota_f \rangle$ *where*

- $[f_T] : TG_1 \leftrightarrow TG_2$ *is a semi-abstract span in* **Graph***, called the* type-span*;*
- $f_P : P_1 \to P_2 \cup \{\emptyset\}$ *is a total function, where* $\emptyset$ *is a new production name (not in* P_2*), with associated production* $\emptyset \rightarrowtail \emptyset$*;*[4]
- ι_f *is a family* $\{\iota_f(q_1) \mid q_1 \in P_1\} \cup \{\iota_f^s\}$ *of morphisms in* **Graph** *such that* $\iota_f^s : |G_{s_2}| \to |G_{s_1}|$ *and for each* $q_1 \in P_1$*, if* $f_P(q_1) = q_2$*, then* $\iota_f(q_1)$ *is pair*

$$\langle \iota_f^L(q_1) : |L_{q_2}| \to |L_{q_1}|, \iota_f^R(q_1) : |R_{q_2}| \to |R_{q_1}| \rangle.$$

[4] Considering the empty production $\emptyset$ is technically preferable to the use of a partial mapping $f_P : P_1 \rightarrowtail P_2$.

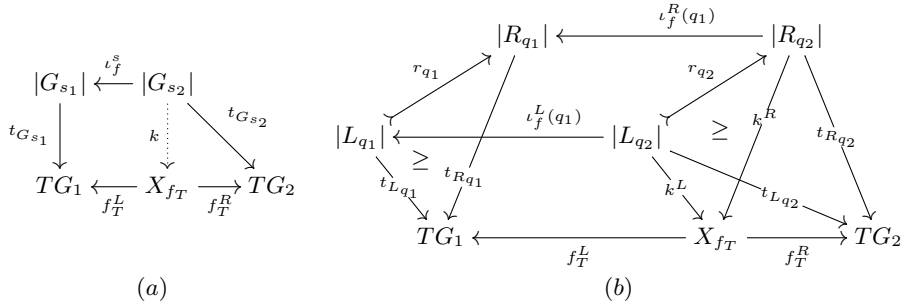


Fig. 7. Diagrams for SPO grammar morphisms.

such that the following conditions are satisfied:

1. Preservation of the start graph.
   There exists a morphism k such that $f_T\{\iota_f^s, k\}(G_{s_1}, G_{s_2})$, namely such that the diagram in Fig. 7.(a) commutes and the square is a pullback.
2. Preservation of productions.
   For each $q_1 \in P_1$, with $q_2 = f_P(q_1)$, there exist morphisms k^L and k^R such that the diagram in Fig. 7.(b) commutes, and $f_T\{\iota_f^Y(q_1), k^Y\}(Y_{q_1}, Y_{q_2})$ for $Y \in \{L, R\}$.

The morphism f is called relational *if the type component f_T is relational.*

As in [1, 7] one can show that grammar morphisms are "simulations", namely that every derivation ρ_1 in $\mathcal{G}_1$ can be transformed into a derivation ρ_2 in $\mathcal{G}_2$, related to ρ_1 by the pullback-retyping construction induced by the morphism.

2 Nondeterministic occurrence grammars

Analogously to what happens for Petri nets, occurrence grammars are "safe" grammars, where the dependency relations between productions satisfy suitable acyclicity and well-foundedness requirements. Nondeterministic occurrence grammars will be used to provide a static description of the computation of a given graph grammar, recording the events (production applications) which can appear in all possible derivations and the dependency relations among them.

While for nets it suffices to take into account only the causality and conflict relations, for grammars 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 between productions. Quite interestingly, instead, as we shall discuss later there is no need of taking into account the dependencies between events related to the side-effects of rule applications (i.e., the deletion of an edge caused by the deletion of its source or target node).

The notion of safe graph grammar [8] generalises the one for P/T nets which requires that each place contains at most one token in any reachable marking.

Definition 10 ((strongly) safe grammar). *A grammar* $\mathcal{G} = \langle TG, G_s, P, \pi \rangle$ *is* (strongly) safe *if, for all* H *such that* $G_s \Rightarrow^* H$, H *is injective.*

In a safe grammar, each graph G reachable from the start graph is injectively typed, and thus we can identify it with the corresponding subgraph $t_G(|G|)$ of the type graph. With this identification, 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. Thus, according to its typing, we can safely think that a production *produces*, *preserves* or *consumes* 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. For instance, for grammar $\mathcal{G}$ in Fig. 8, ${}^\bullet q_1 = \{A\}$, $\underline{q_1} = \{B\}$ and $q_1{}^\bullet = \{L\}$, while ${}^\bullet B = \emptyset$, $\underline{B} = \{q_1, q_2\}$ and $B^\bullet = \{q_3\}$.

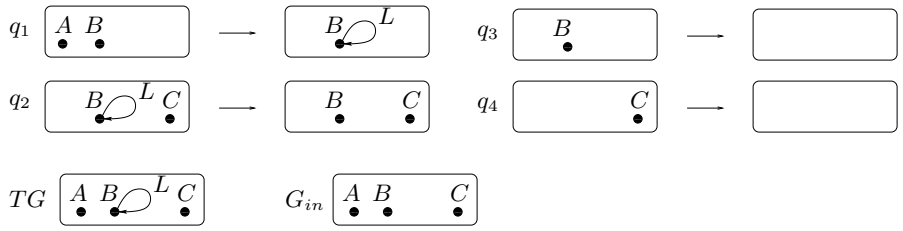


Fig. 8. A safe SPO graph grammar $\mathcal{G}$.

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 11 (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 edge* x *in the type graph* TG*, and for productions* $q, q' \in P$

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

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* $\{q \in P : q \leq x\}$*.*

Notice that the fact that an item is preserved by q and consumed by q', i.e., $\underline{q} \cap {}^\bullet q' \neq \emptyset$ (e.g., item $C \in \underline{q_2} \cap {}^\bullet q_4$ in grammar $\mathcal{G}$ of Fig. 8), does not imply $q < q'$. Actually, the dependency between the two productions is a kind of *asymmetric conflict* (see [2, 23, 17]). The application of q' prevents q from being applied, so that q can never follow q' in a derivation (or equivalently when both q and q' occur in a derivation then q must precede q'). But the converse is not true, since q *can* be applied before q'.

Definition 12 (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 $q \cap {}^\bullet q' \neq \emptyset$ then $q \nearrow q'$;*
2. *if ${}^\bullet q \cap {}^\bullet q' \neq \emptyset$ and $q \neq q'$ then $q \nearrow q'$;*
3. *if $q < q'$ then $q \nearrow q'$.*

Condition 1 is justified by the discussion above. Condition 2 essentially expresses the fact that the ordinary symmetric conflict is encoded, 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).

As already mentioned, the side-effects of production applications can be disregarded when analysing the dependency relations between events. In fact:

Causality. Assume that production q produces an edge e, and q' deletes e as side-effect (because it deletes its source or target). At a first glance we could think that q' should causally depend on q. However, although q' consumes the resource e produced by q, the application of q is not necessary to make q' applicable, since q' does not explicitly require the presence of e. Hence q' does not causally depend on q. For instance, referring to grammar $\mathcal{G}$ in Fig. 8, the application of q_3 after q_1 deletes node B and edge L as side-effect. However q_3 does not depend on q_1 since it can be applied already to the start graph.

Asymmetric conflict. Also asymmetric conflict (called *weak conflict* in [24]) can be defined disregarding the mentioned side-effects. This is basically due to the fact that when a production uses (consumes or preserves) an edge, it must use necessarily the corresponding source and target nodes, and therefore dependencies related to side-effects can be detected by looking only at explicitly used items. E.g., consider again grammar $\mathcal{G}$ in Fig. 8. Observe that production q_3 prevents q_2 from being applied since it deletes, as side-effect, edge L which is consumed by q_2. However, to consume L, production q_2 must preserve or consume node B (actually, it consumes it) and thus the "ordinary" definition of asymmetric conflict already tells us that $q_2 \nearrow q_3$.

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

Definition 13 ((nondeterministic) occurrence grammar). *A* (nondeterministic) occurrence grammar *is a grammar $\mathcal{O} = \langle TG, G_s, P, \pi \rangle$ 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 the asymmetric conflict $\nearrow$ is acyclic on $\lfloor q \rfloor$;*
2. *the start graph G_s is 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. *any item x in TG is created by at most one production in P, namely $| {}^\bullet x | \leq 1$;*

4. *for each* $q \in P$, *the typing* t_{L_q} *is injective on the "consumed part"* $|L_q| - |dom(r_q)|$, *and* t_{R_q} *is injective on the "produced part"* $|R_q| - r_q(|dom(r_q)|)$.

We denote by **OGG** *the full subcategory of* **GG** *with occurrence grammars as objects.*

Since the start 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 [4]). In particular, in Condition (1), the acyclicity of asymmetric conflict on $\lfloor q \rfloor$ corresponds to the requirement of irreflexivity for the conflict relation in occurrence nets. Condition (4), instead, is closely related to safety and requires that each production consumes and produces items with multiplicity one. An example of occurrence grammar is $\mathcal{G}$ in Fig. 8.

As in the case of Petri nets, reachable states can be characterised in terms of a concurrency relation.

Definition 14 (concurrent graph). *Let* $\mathcal{O} = \langle TG, P, \pi \rangle$ *be an occurrence grammar. A subgraph* G *of* TG *is called* concurrent *if*

1. $\nearrow_G$, *the asymmetric conflict restricted to* $\bigcup_{x \in G} \lfloor x \rfloor$, *is acyclic and finitary;*
2. $\neg(x < y)$ *for all* $x, y \in G$.

It is possible to show that a subgraph G of TG is concurrent iff it is a subgraph of a graph reachable from the start graph by means of a derivation which applies all the productions in $\bigcup_{x \in G} \lfloor x \rfloor$ exactly once in any order compatible with $\nearrow$.

3 Unfolding of graph grammars

The unfolding construction, when applied to a consuming grammar $\mathcal{G}$, produces a nondeterministic occurrence grammar $\mathcal{U}_s(\mathcal{G})$ describing the behaviour of $\mathcal{G}$. The unfolding can be characterised as a universal construction for several interesting categories of algebraic graph grammars.

Intuitively, given a graph grammar $\mathcal{G}$, the construction consists of starting from the start graph of $\mathcal{G}$, then applying in all possible ways its productions to concurrent subgraphs, 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. Due to space limitations we skip the details of the constructions, giving only a summary of the main results.

3.1 Unfolding of semi-weighted graph grammars

As it has been done for ordinary (and other larger classes of) Petri nets [27, 20, 1], we first restrict to a full subcategory **SGG** of **GG** where objects satisfy conditions analogous to those defining semi-weighted P/T Petri nets. A graph

grammar is semi-weighted if the start graph is injective and the right-hand side of each production is injective when restricted to produced items (namely, to items which are not in the codomain of the production morphism).

Theorem 1. *The unfolding construction can be expressed as a functor* $\mathcal{U}_s : \mathbf{SGG} \to \mathbf{OGG}$*, which is right adjoint to the inclusion* $\mathcal{I}_s : \mathbf{OGG} \to \mathbf{SGG}$*.*

3.2 Unfolding of general grammars

The restriction to the semi-weighted case is essential for the universal characterisation of the unfolding construction when one uses general morphisms. However, suitably restricting graph grammar morphisms to still interesting subclasses (comprising, for instance, the morphisms of [24, 14]) it is possible to regain the categorical result for general, possibly non semi-weighted, grammars.

More specifically, the coreflection result can be obtained by limiting our attention to a (non full) subcategory $\widehat{\mathbf{GG}}$ of $\mathbf{GG}$, where objects are general graph grammars, but all morphisms have a relational span as type component. The naive solution of taking *all* relational morphisms as arrows of $\widehat{\mathbf{GG}}$ does not work because they are not closed under composition. A possible appropriate choice is instead given by the category $\mathbf{GG}^R$, where the arrows are grammar morphisms such that the *right* component of the type span is mono. It is easy to realize that these kinds of span corresponds to partial graph morphisms in the opposite direction. In fact, a partial graph morphism $g : TG_2 \rightarrowtail TG_1$ can be identified with the span

$$TG_1 \xleftarrow[g]{} dom(g) \hookrightarrow TG_2$$

Theorem 2. *The unfolding construction can be turned into a functor* $\mathcal{U}_s^R : \mathbf{GG}^R \to \mathbf{OGG}^R$*, having the inclusion* $\mathcal{I}_s^R : \mathbf{OGG}^R \to \mathbf{GG}^R$ *as left adjoint, establishing a coreflection between the two categories.*

Alternatively, the result can be proved for the subcategory $\mathbf{GG}^L$ of $\mathbf{GG}$ where arrows are grammar morphisms having the *left* component of the type span which is mono (corresponding to partial graph morphisms with the same source and target of the span).

4 Event structure semantics for SPO graph grammars

In this section we show that asymmetric event structures, a generalisation of prime event structures introduced in [4], provide a suitable setting for defining an event structure semantics for SPO graph grammars. After reviewing the basics of asymmetric event structures, we show that any occurrence SPO grammar can be mapped to an asymmetric event structure via a functorial construction. Furthermore, a left adjoint functor, back from asymmetric event structures to occurrence grammars, can be defined, associating a canonical occurrence grammar to any asymmetric event structure.

4.1 Asymmetric event structures

Asymmetric event structures [4] are a generalisation of prime event structures where the conflict relation is allowed to be non-symmetric. As already mentioned, this is needed to give a faithful representation of dependencies between events in formalisms such as string, term, graph rewriting and contextual nets, where a rule may preserve a part of the state, in the sense that part of the state is necessary for applying the rule, but it is not affected by the application.

For technical reasons we first introduce pre-asymmetric event structures. Then asymmetric event structures will be defined as special pre-asymmetric event structures satisfying a suitable condition of "saturation".

Definition 15 (asymmetric event structure). *A* pre-asymmetric event structure (pre-AES) *is a tuple* $\mathcal{A} = \langle E, \leq, \nearrow \rangle$*, where* E *is a set of* events *and* $\leq$*,* $\nearrow$ *are binary relations on* E *called* causality *and* asymmetric conflict*, respectively, such that:*

1. $\leq$ *is a partial order and* $\lfloor e \rfloor = \{e' \in E \mid e' \leq e\}$ *is finite for all* $e \in E$*;*
2. $\nearrow$ *satisfies, for all* $e, e' \in E$*:*

$$\text{(a) } e < e' \quad \Rightarrow \quad e \nearrow e', \qquad\qquad \text{(b) } \nearrow \text{ is acyclic in } \lfloor e \rfloor,$$

where, as usual, $e < e'$ *means* $e \leq e'$ *and* $e \neq e'$*.*

An asymmetric event structure *(*AES*) is a pre-*AES *which satisfies:*

3. *for any* $e, e' \in E$*, if* $\nearrow$ *is cyclic in* $\lfloor e \rfloor \cup \lfloor e' \rfloor$ *then* $e \nearrow e'$*.*

The asymmetric conflict relation $\nearrow$ determines an order of execution locally to each computation: if $e \nearrow e'$ and e, e' occur in the same computation then e must precede e'. Therefore a set of events $e_1 \nearrow e_2 \nearrow \ldots \nearrow e_n \nearrow e_1$ forming a cycle of asymmetric conflict can never occur in the same computation, a fact that can be naturally interpreted as a kind of conflict over sets of events. Condition (3) above ensures that, in an AES, this kind conflict is inherited through causality, a typical property also of PES's.

Any pre-AES can be "saturated" to produce an AES. More precisely, given a pre-AES $\mathcal{A} = \langle E, \leq, \nearrow \rangle$, its saturation, denoted by $\overline{\mathcal{A}}$, is the AES $\langle E, \leq, \nearrow' \rangle$, where $\nearrow'$ is defined as $e \nearrow' e'$ iff $(e \nearrow e')$ or $\nearrow$ is cyclic in $\lfloor e \rfloor \cup \lfloor e' \rfloor$.

Definition 16 (category of AES's). *Let* $\mathcal{A}_0$ *and* $\mathcal{A}_1$ *be two* AES*'s. An* AES-morphism $f : \mathcal{A}_0 \to \mathcal{A}_1$ *is a partial function* $f : E_0 \rightarrowtail E_1$ *such that, for all* $e_0, e_0' \in E_0$*, assuming that* $f(e_0)$ *and* $f(e_0')$ *are defined,*

1. $\lfloor f(e_0) \rfloor \subseteq f(\lfloor e_0 \rfloor)$*;*
2. *(a)* $f(e_0) \nearrow_1 f(e_0') \quad \Rightarrow \quad e_0 \nearrow_0 e_0'$*;*
   (b) $(f(e_0) = f(e_0')) \wedge (e_0 \neq e_0') \quad \Rightarrow \quad e_0 \nearrow e_0'$*.*

We denote by **AES** *the category having asymmetric event structures as objects and* AES*-morphisms as arrows.*

The notion of configuration extends smoothly from PES's to AES's, the main difference being the fact that the computational order between configurations is not simply set-inclusion. In fact, a configuration C can be extended with an event e' only if for any event $e \in C$, it does not hold that $e' \nearrow e$ (since, in this case, e would disable e'). The set of configurations of an AES with such a computational order is a domain. The corresponding functor from **AES** to **Dom**, the category of finitary prime algebraic domains, has a left adjoint which maps each domain to the corresponding prime event structure (each PES can be seen as a special AES where conflict is symmetric). Hence Winskel's equivalence between **PES**, the category of prime event structures, and **Dom** generalises to a coreflection between **AES** and **Dom**.

$$\mathbf{AES} \underset{\mathcal{L}_a}{\overset{\mathcal{P}_a}{\underset{\longrightarrow}{\overset{\longleftarrow}{\perp}}}} \mathbf{Dom}$$

4.2 From occurrence grammars to AES's

Given any occurrence grammar, the corresponding asymmetric event structure is readily obtained by taking the production names as events. Causality and asymmetric conflict are the relations defined in Definitions 11 and 12.

Definition 17 (AES for an occurrence grammar). *Let $\mathcal{O} = \langle TG, P, \pi \rangle$ be an occurrence grammar. The* AES *associated to $\mathcal{O}$, denoted $\mathcal{E}_s(\mathcal{O})$, is the saturation of the pre-*AES *$\langle P, \leq, \nearrow \rangle$, with $\leq$ and $\nearrow$ as in Definitions 11 and 12.*

The above construction naturally gives rise to a functor.

Proposition 1. *For any morphism $h : \mathcal{O}_0 \to \mathcal{O}_1$ between occurrence grammars, let $\mathcal{E}_s(h)(q) = h_P(q)$ if $h_P(q) \neq \emptyset$ and $\mathcal{E}_s(h)(q)$ undefined, otherwise. Then $\mathcal{E}_s : \mathbf{OGG} \to \mathbf{AES}$ is a well-defined functor.*

For instance, Fig. 9 shows the AES (and the prime algebraic domain of its configurations) associated to the occurrence grammar $\mathcal{G}$ in Fig. 8. In the AES straight and dotted arrows represent causality and asymmetric conflict, respectively. In any configuration the event corresponding to q_i is written as "i".

4.3 From AES's to occurrence grammars

Any AES is identified with a canonical occurrence grammar, via a free construction that mimics Winskel's one. Given an asymmetric event structure $\mathcal{A}$, the corresponding grammar has the events of $\mathcal{A}$ as production names, while the graph items are freely generated in order to induce the right kind of dependencies between events. More specifically, first the graph nodes are freely generated according to the dependencies in $\mathcal{A}$. Then for any pair of nodes, edges connecting the two nodes are freely generated according to the dependencies in $\mathcal{A}$ and the specific restrictions of the SPO rewriting mechanism.

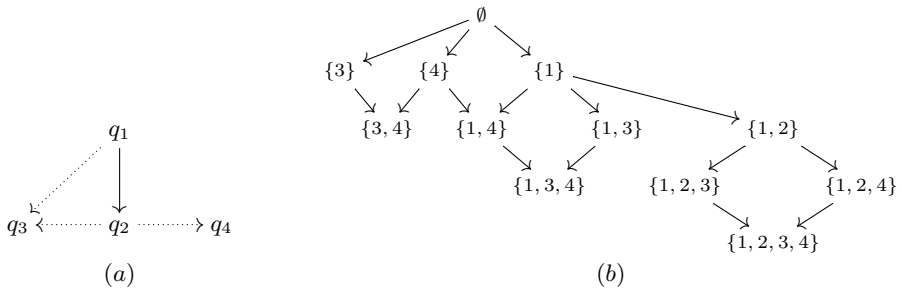


Fig. 9. The (a) AES and (b) domain of configurations for $\mathcal{G}$ of Fig. 8.

Definition 18. *Let $\mathcal{A} = \langle Ev, \leq, \nearrow \rangle$ be an* AES*. The corresponding* SPO *occurrence graph grammar, denoted by $\mathcal{N}_s(\mathcal{A}) = \langle TG, P, \pi \rangle$, is defined as follows:*

- *The* type graph *$TG = \langle N, E, s, t \rangle$ is defined as below, where $A, B, \ldots$ range over generic sets of events and x over sets of events of cardinality at most* 1 *(singletons or the empty set). Moreover by $x < e$, if $x = \{e'\}$ we mean that $e' < e$, while the relation trivially holds if $x = \emptyset$ (i.e. $\emptyset < e$, for any event e). Symmetrically, by $e < x$ with $x = \{e'\}$ we mean $e < e'$, while $e < \emptyset$ is intended to be always false.*
  - *Nodes:*
    $$N = \left\{ \langle x, A, B \rangle : \begin{array}{c} \forall e \in A \cup B.\ x < e, \\ \forall a \in A.\ \forall b \in B.\ a \nearrow b, \\ \forall b, b' \in B.\ b \neq b' \Rightarrow b \nearrow b' \end{array} \right\};$$
  - *Edges:*
    $$E = \left\{ \langle x, A, B, n_1, n_2 \rangle : \begin{array}{c} n_i = \langle x_i, A_i, B_i \rangle \in N, \\ \forall e \in A \cup B.\ x < e, \\ \forall a \in A.\ \forall b \in B.\ a \nearrow b, \\ \forall b, b' \in B.\ b \neq b' \Rightarrow b \nearrow b' \\ x_i \leq x \text{ for } i \in \{1, 2\} \\ A \subseteq A_1 \cap A_2 \\ B \subseteq (A_1 \cup B_1) \cap (A_2 \cup B_2) \\ \forall e_i \in B_i.\ \neg(e_i \leq x_j) \text{ for } i, j \in \{1, 2\},\ i \neq j \end{array} \right\};$$
  - *Source and target functions:*
    $$s(\langle x, A, B, n_1, n_2 \rangle) = n_1 \quad \text{and} \quad t(\langle x, A, B, n_1, n_2 \rangle) = n_2.$$
- *The set of productions $P = Ev$, and for any event $e \in Ev$ the corresponding production $\pi(e) = L_e \rightarrowtail R_e$ is defined as follows:*
  - $|L_e| = \{n = \langle x, A, B \rangle,\ l = \langle x, A, B, n_1, n_2 \rangle \mid e \in A \cup B\}$
  - $|R_e| = \{n = \langle x, A, B \rangle,\ l = \langle x, A, B, n_1, n_2 \rangle \mid e \in x \cup A\}$

  The typing and the (partial) inclusion of L_e in R_e are the obvious ones.

A node in the type graph TG is a triple $n = \langle x, A, B\rangle$. The set x might contain the event which generates the node n or might be empty if the node is in the start graph, A is the set of events which preserve the node n and B is the set of events which consume n. Clearly, the event in x, if any, must be a cause for every event in $A \cup B$, the events in A must be in asymmetric conflict with the events in B, and the events in B must be pairwise in conflict (represented as an asymmetric conflict in both directions, i.e., $b \nearrow b'$ and $b' \nearrow b$).

An edge in the type graph is a tuple $l = \langle x, A, B, n_1, n_2\rangle$. The meaning of x, A, B is the same as for nodes. The components n_1 and n_2 are intended to represent the source and target nodes of edge l. They are subject to requirements which arise from the specific features of the SPO rewriting mechanism. First, $x_i \leq x$ since the event which produces an edge must produce or preserve the source/target nodes. Any event which preserves the edge must also preserve the source/target, hence $A \subseteq A_1 \cap A_2$. Any event which consumes the edge must preserve or consume the source/target nodes, hence $B \subseteq (A_1 \cup B_1) \cap (A_2 \cup B_2)$.

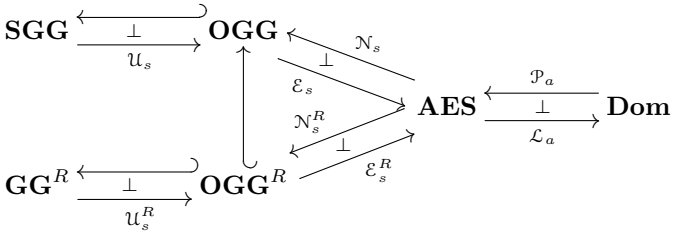
Finally the nodes n_1 and n_2 must be allowed to coexist: the requirement $x_i \leq x$ already ensures that n_1 and n_2 are not in conflict. Moreover each node is asked not to causally depend on the events which *consume* the other one.

We conclude with the main result, stating that the construction of the occurrence grammar associated to an AES is functorial and left adjoint to $\mathcal{E}_s$, establishing a coreflection between **OGG** and **AES**. For any AES $\mathcal{A}$, $\mathcal{E}_s(\mathcal{N}_s(\mathcal{A})) = \mathcal{A}$ and the component at $\mathcal{A}$ of the unit of the adjunction is the identity.

Theorem 3 (coreflection between OGG and AES). *The construction $\mathcal{N}_s$ extends to a functor that is left adjoint to $\mathcal{E}_s$.*

Roughly speaking, the proof shows that, given any AES $\mathcal{A}$ and occurrence graph grammar $\mathcal{O}$, all AES-morphisms $f : \mathcal{A} \to \mathcal{E}_s(\mathcal{O})$ uniquely extends to graph grammar morphisms $\hat{f} : \mathcal{N}_s(\mathcal{A}) \to \mathcal{O}$. The type span component of morphism $\hat{f}$ is $TG_{\mathcal{N}_s(\mathcal{A})} \stackrel{f_T}{\leftarrow} TG_{\mathcal{O}} \stackrel{id}{\rightarrow} TG_{\mathcal{O}}$, where f_T maps any item in $TG_{\mathcal{O}}$ to the only item in $TG_{\mathcal{N}_s(\mathcal{A})}$ which induces analogous dependencies among the events.

Summing up, Theorem 1 and Theorem 3 above give a chain of coreflections from the category **SGG** of semi-weighted SPO graph grammars to **AES** and **Dom**. The result can be extended to $\mathbf{GG}^R$, the category of general SPO grammars with restricted morphisms (having the *right* component in the type span which is mono), by exploiting Theorem 2 and observing that $\mathcal{N}_s$ restricts to a well-defined functor $\mathcal{N}_s^R : \mathbf{AES} \to \mathbf{OGG}^R$. The possibility of generalising the result to other categories of grammars with relational morphisms is still open.



5 Conclusions

We have defined a functorial concurrent semantics for SPO graph grammars, expressed as a chain of coreflections leading from various categories of SPO grammars to the categories of AES's and domains. The approach originally proposed by Winskel in the setting of Petri nets has been fully extended to SPO graph grammars, improving the previous proposals where some steps of the construction were lacking, notably, in the case of the DPO approach, the functor from event structures to occurrence grammars.

A natural question regards the possibility of using these results for the DPO approach. We have already mentioned that for DPO graph grammars, due to the presence of application conditions for rules, a more complex kind of event structures, called *inhibitor event structures* [1] was introduced to obtain a functorial semantics. In this way a functor mapping any occurrence DPO grammar to an IES can be defined, which, however it does not admit a left adjoint. Still an idea could be to view asymmetric event structures as a coreflective subcategory of inhibitor event structures and then to devise a construction which associates a canonical DPO grammar to any asymmetric event structure.

The theory developed in this paper naturally suggests a notion of graph process for SPO grammars, which can be defined as a deterministic occurrence grammar with a morphism to the original grammar. We conjecture that these processes correspond exactly to the *concurrent derivations* of [16], which in turn were characterised as special classes of graph grammars in [24].

The analogies between the first steps of the constructions for the SPO and DPO approaches (the proper unfolding constructions) suggest the possibility of developing a general theory of unfolding in abstract categories (e.g., high level replacement systems [10]). Some parts of the construction are rather concrete and not easy to recast in an abstract categorical setting, but still this represents a challenging topic of further investigation.

References

1. P. Baldan. *Modelling concurrent computations: from contextual Petri nets to graph grammars.* PhD thesis, Department of Computer Science, University of Pisa, 2000. Available as technical report n. TD-1/00.
2. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. *Proc. of FoSSaCS '98*, vol. 1378 of *LNCS*, pp. 63–80. Springer, 1998.
3. P. Baldan, A. Corradini, and U. Montanari. Unfolding of double-pushout graph grammars is a coreflection. *TAGT'98 Conference Proc.*, vol. 1764 of *LNCS*, pp. 145–163. Springer, 1999.
4. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.
5. S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. *Applications and Theory of Petri Nets*, vol. 691 of *LNCS*, pp. 186–205. Springer, 1993.

6. A. Corradini. Concurrent graph and term graph rewriting. *Proc. of CONCUR'96*, vol. 1119 of *LNCS*, pp. 438–464. Springer, 1996.
7. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. The category of typed graph grammars and its adjunctions with categories of derivations. *Proc. of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, vol. 1073 of *LNCS*. Springer, 1996.
8. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
9. 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 Rozenberg [25], chapter 3.
10. H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in High-Level Replacement Systems. *Mathematical Structures in Computer Science*, 1:361–404, 1991.
11. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic Approaches to Graph Transformation II: Single Pushout Approach and Comparison with Double Pushout Approach. In Rozenberg [25], chapter 4.
12. H. Ehrig, M. Pfender, and H.J. Schneider. Graph-grammars: an algebraic approach. In *Proc. of IEEE Conf. on Automata and Switching Theory*, pp. 167–180, 1973.
13. U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
14. R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and vertical structuring of graph transformation systems. *Mathematical Structures in Computer Science*, 6(6):613–648, 1996.
15. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.
16. M. Korff. *Generalized graph structure grammars with applications to concurrent object-oriented systems*. PhD thesis, Technische Universität Berlin, 1996.
17. R. Langerak. *Transformation and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.
18. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.
19. M. Löwe, M. Korff, and A. Wagner. An Algebraic Framework for the Transformation of Attributed Graphs. *Term Graph Rewriting: Theory and Practice*, pp. 185–199. Wiley, London, 1993.
20. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Place/Transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1997.
21. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6), 1995.
22. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
23. G. M. Pinna and A. Poigné. On the nature of events: another perspective in concurrency. *Theoretical Computer Science*, 138(2):425–454, 1995.
24. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.
25. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*. World Scientific, 1997.
26. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proc. of ICALP'97*, vol. 1256 of *LNCS*, pp. 538–548. Springer, 1997.
27. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, vol. 255 of *LNCS*, pp. 325–392. Springer, 1987.