Unfolding of Double-Pushout Graph Grammars is a Coreflection*

Paolo Baldan, Andrea Corradini, and Ugo Montanari

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

Abstract. In a recent paper, mimicking Winskel's construction for Petri nets, a concurrent semantics for (*double-pushout*) *DPO graph grammars* has been provided by showing that each graph grammar can be unfolded into an acyclic branching structure, that is itself a (nondeterministic occurrence) graph grammar describing all the possible computations of the original grammar.
This paper faces the problem of providing a closer correspondence with Winskel's result by showing that the unfolding construction can be described as a coreflection between the category of graph grammars and the category of occurrence graph grammars. The result is shown to hold for a suitable subclass of graph grammars, called *semi-weighted graph grammars*. Unfortunately the coreflection does not extend to the whole category of graph grammars: some ideas for solving the problem are suggested.

1 Introduction

In recent years, various concurrent semantics for graph rewriting systems have been proposed in the literature, some of which are inspired by their correspondence with Petri nets (see [5] 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 [18], shows that the event structure semantics of *safe* nets can be given via a chain of coreflections starting from the category **Safe** of safe nets, through category **Occ** of occurrence nets. The event structure associated with a net is obtained by first constructing a "nondeterministic unfolding" of the net, and then by considering only its transitions and the causal and conflict relations among them. In [14, 15] it is shown that essentially the same constructions work for the larger category of *semi-weighted nets*, i.e., P/T nets where the initial marking is a set and transitions can generate at most one token in each post-condition. Winskel's result has been also extended, in [2], to a more general class of nets called (semi-weighted) contextual nets or nets with read (test) arcs. Contextual nets generalize classical nets by adding the possibility of checking for the presence of a token in a place, without consuming it. Their capability of "preserving part" of the state in a rewriting step makes this kind of nets closer to graph grammars. Indeed, starting from these results, the paper [3] shows that a Winskel's style construction

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

allows one to unfold each graph grammar into a nondeterministic occurrence grammar describing its behaviour. The unfolding is used to define a prime algebraic domain and an event structure semantics for the grammar.

In this paper we make a further step towards full correspondence with Winskel's result by facing the problem of characterizing the unfolding construction for DPO graph grammars just mentioned as a true coreflection.

Section 2 reviews the basics of DPO typed graph grammars and introduces the notion of grammar morphism, a slight variation of the morphisms in [6], making the class of graph grammars a category **GG**. Section 3 recalls the notion of *nondeterministic occurrence grammar* [3], which are grammars satisfying suitable acyclicity and well-foundedness requirements, representing in a unique "branching" structure several possible "acyclic" grammar computations. The full subcategory of **GG** having occurrence grammars as objects is denoted by **OGG**. By exploiting the notions of occurrence grammar and of grammar morphism, Section 4 defines *nondeterministic graph process*. As in Petri net theory, a nondeterministic process of a grammar $\mathcal{G}$ consists of a (suitable) grammar morphism from an occurrence grammar to $\mathcal{G}$. Nicely, deterministic finite processes turn out to coincide with the graph processes of [1].

Section 5 presents the 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 unfolding is endowed with a morphism $\chi_{\mathcal{G}}$ into the original grammar $\mathcal{G}$, making $\mathcal{U}(\mathcal{G})$ a process of $\mathcal{G}$. Next, Section 6 faces the problem of turning the unfolding construction into a functor establishing a coreflection between the categories of graph grammars and of occurrence grammars. As in the case of Petri nets we restrict to those grammars where the initial graph and the items produced by each production are injectively typed. Such grammars, by analogy with the corresponding subclass of Petri nets, are called *semi-weighted*, and the corresponding full subcategory of **GG** is denoted by **SGG**. We show that the unfolding construction extends to a functor $\mathcal{U} : \mathbf{SGG} \to \mathbf{OGG}$ which is right adjoint to the inclusion $\mathcal{I}_O : \mathbf{OGG} \to \mathbf{SGG}$, and thus establishes a coreflection between the two categories.

In Section 7, we show that unfortunately the result cannot be extended in a trivial way to the whole category **GG** of graph grammars. Even worse, a counterexample shows that there is no way of turning the unfolding construction into a functor which is right adjoint to the inclusion $\mathcal{I} : \mathbf{OGG} \to \mathbf{GG}$. Starting from this negative result some possible ways of solving the problem are singled out.

Because of space limitations we are forced to defer to the full version the detailed comparison with the related work in the literature, comprising different notions of graph grammar morphisms [6, 12, 17, 4] as well as the unfolding construction for SPO grammars in [17]. For the same reason also the proofs of our statements are omitted.

2 Typed graph grammars and their morphisms

This section first summarizes the basic definitions about typed graph grammars [8], a variation of classical DPO graph grammars [10, 9] which uses *typed graphs*, namely graphs labelled over a structure (the *graph of types*) that is itself a graph. Next some insights are given on the relationship between typed graph grammars and Petri nets.

Finally, the class of typed graph grammars is turned into a category **GG** by introducing a notion of grammar morphism.

2.1 Typed graph grammars

Let **Graph** be the category of (directed, unlabelled) graphs and total graph morphisms. For a graph G we will denote by N_G and E_G the sets of *nodes* and *arcs* of G, and by $s_G, t_G : E_G \to N_G$ its *source* and *target* functions. Given a graph TG, a *typed graph* G over TG is a graph $|G|$, together with a morphism $t_G : |G| \to TG$. A morphism between TG-typed graphs $f : G_1 \to G_2$ is a graph morphisms $f : |G_1| \to |G_2|$ consistent with the typing, i.e., such that $t_{G_1} = t_{G_2} \circ f$. A typed graph G is called *injective* if the typing morphism t_G is injective. The category of TG-typed graphs and typed graph morphisms is denoted by TG-**Graph** and can be sinthetically defined as the comma category $(\mathbf{Graph} \downarrow TG)$.

Fixed a graph TG of types, a *(TG-typed graph) production* $(L \overset{l}{\leftarrow} K \overset{r}{\to} R)$ is a pair of *injective* typed graph morphisms $l : K \to L$ and $r : K \to R$, where $|L|$, $|K|$ and $|R|$ are finite graphs. 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.

Definition 1 (typed graph grammar). *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 *is a function which associates a graph production to each production name in* P*.*

We denote by $\mathit{Elem}(\mathcal{G})$ the set $N_{TG} \cup E_{TG} \cup P$. Furthermore, we will assume that for each production name q the corresponding production $\pi(q)$ is $L_q \overset{l_q}{\leftarrow} K_q \overset{r_q}{\to} R_q$, where, without loss of generality, the injective morphisms l_q and r_q are inclusions.

Since in this paper we work only with typed notions, we will usually omit the qualification "typed", and, sometimes, 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.

Definition 2 (direct derivation). *Given a typed graph* G*, a production* q*, and a* match *(i.e., a graph morphism)* $g : L_q \to G$*, 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*

$$\begin{tikzcd}
q : L_q \arrow[d, "g"'] & K_q \arrow[l, "l_q"'] \arrow[r, "r_q"] \arrow[d, "k"] & R_q \arrow[d, "h"] \\
G & D \arrow[l, "b"] \arrow[r, "d"'] & H
\end{tikzcd}$$

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

Given an injective morphism $l_q : K_q \to L_q$ and a match $g : L_q \to G$ 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) exists if and only if the *gluing condition* is satisfied. This consists of two parts:

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

A *derivation* over a grammar $\mathcal{G}$ is a sequence of direct derivations (over $\mathcal{G}$) starting from the initial graph, namely $\rho = \{G_{i-1} \Rightarrow_{q_{i-1}} G_i\}_{i \in \{1,\ldots,n\}}$, with $G_0 = G_{in}$.

2.2 Relation with Petri nets.

The notion of grammar morphism, and many definitions and constructions in this paper are better understood keeping in mind the relation between Petri nets and DPO graph grammars. The basic observation (which belongs to the folklore, see, e.g., [5]) 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 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.

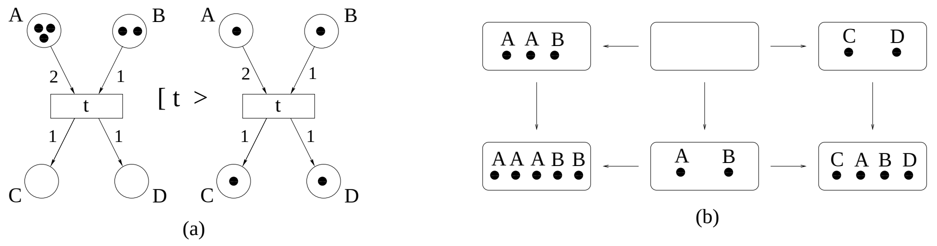

Fig. 1. Firing of a transition and corresponding DPO direct derivation.

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).

2.3 Grammar Morphisms

The notion of grammar morphism we are going to introduce is very similar to the one originally defined in [6], which was in turn introduced as a generalization of Petri nets morphisms. Recall that a Petri net morphism [18] 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 pre-set and post-set of transitions, 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.

Since the items of the graph of types of a grammar can be seen as a generalization of Petri net places, the first component of a grammar morphism will be a span between the type graphs of the source and target grammars, arising as a categorical generalization of the notion of multirelation. For an extensive discussion of this idea we refer the reader to [6, 4]. The following definitions will be useful.

Definition 3 (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 an the target of the span and we will write* $f : a \leftrightarrow b$*. The span* f *will be sometimes written as* $\langle f^L, x_f, f^R \rangle$*, explicitly giving the common source object* x_f*.*

Consider now 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. 2.(a)). The isomorphism class of a span* f *will be denoted by* $[f]$ *and called a* semi-abstract span.

Definition 4 (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. 2.(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* $\mathbf{C}$*.*

It can be shown that composition is well-defined, namely it does not depend on the particular choice of the representatives, and that it is associative.

$$\begin{tikzcd}
 & x_{f'} \arrow[dl, "f'^L"'] \arrow[dr, "f'^R"] & \\
a & x_f \arrow[l, "f^L"] \arrow[r, "f_R"'] \arrow[u, "k"] & b
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & & & x_f \arrow[dll, dotted, "y"'] \arrow[dr, dotted, "z"] & & \\
a & x_{f_1} \arrow[l, "f_1^L"] \arrow[rr, "f_1^R"'] & & b & x_{f_2} \arrow[l, "f_2^L"] \arrow[r, "f_2^R"'] & c
\end{tikzcd}$$

(b)

Fig. 2. Equivalence and composition of spans.

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graph grammars and let $[f_T] : TG_1 \leftrightarrow TG_2$ be a semi-abstract span between the corresponding type graphs. Observe that $[f_T]$ induces a relation between TG_1-typed graphs and TG_2-typed graphs. In fact, let G_1 be in TG_1-$\mathbf{Graph}$.

Then we can transform G_1 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 $f_T^R : X_{f_T} \to TG_2$.

$$\begin{tikzcd}
{|G_1|} \arrow[d, "t_{G_1}"'] & {|G_2|} \arrow[l, dotted, "x"'] \arrow[d, dotted, "y"] \arrow[dr, "t_{G_2}"] & \\
TG_1 & X_{f_T} \arrow[l, "f_T^L"] \arrow[r, "f_T^R"'] & TG_2
\end{tikzcd}$$

The TG_2-typed graph $G_2 = \langle |G_2|, f_T^R \circ y \rangle$ obtained with this construction, later referred to as *pullback-retyping* construction induced by $[f_T]$, is determined only up to isomorphism. This is due to the definition of pullback and to the fact that, considering semi-abstract spans, we can choose any concrete representative $f'_T \sim f_T$. 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 initial graphs.

Definition 5 (grammar morphism). *Let $\mathcal{G}_i = \langle TG_i, G_{in_i}, P_i, \pi_i \rangle$ ($i \in \{1, 2\}$) 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 $\emptyset \leftarrow \emptyset \to \emptyset$, referred to as the* empty production*;*
- *ι_f is a family $\{\iota_f(q_1) \mid q_1 \in P_1\} \cup \{\iota_f^{in}\}$ such that $\iota_f^{in} : |G_{in_2}| \to |G_{in_1}|$ and for each $q_1 \in P_1$, if $f_P(q_1) = q_2$, then $\iota_f(q_1)$ is triple of morphisms*

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

such that the following conditions are satisfied:

1. Preservation of the initial graph.
There exists a morphism k such that $f_T\{\iota_f^{in}, k\}(G_{in_1}, G_{in_2})$, namely such that the following diagram commutes and the square is a pullback:

$$\begin{tikzcd}
{|G_{in_1}|} \arrow[d, "t_{G_{in_1}}"'] & {|G_{in_2}|} \arrow[l, "\iota_f^{in}"'] \arrow[d, dotted, "k"'] \arrow[dr, "t_{G_{in_2}}"] & \\
TG_1 & X_{f_T} \arrow[l, "f_T^L"] \arrow[r, "f_T^R"'] & TG_2
\end{tikzcd}$$

2. Preservation of productions.
For each $q_1 \in P_1$, with $q_2 = f_P(q_1)$, there exist morphisms k^L, k^K and k^R such that the diagram below commutes, and $f_T\{\iota_f^X(q_1), k^X\}(X_{q_1}, X_{q_2})$ for $X \in \{L, K, R\}$.

$$\begin{tikzcd}
 & & {|R_{q_1}|} \arrow[dddl, "t_{R_{q_1}}"] & & & & {|R_{q_2}|} \arrow[llll, "\iota_f^R(q_1)"'] \arrow[dddr, "t_{R_{q_2}}"] \arrow[dddl, "k^R"] & \\
 & {|K_{q_1}|} \arrow[ur] \arrow[dl] \arrow[dd, "t_{K_{q_1}}"] & & & & {|K_{q_2}|} \arrow[llll, "\iota_f^K(q_1)"'] \arrow[ur] \arrow[dl] \arrow[dd, "k^K"] \arrow[ddrr, "t_{K_{q_2}}"] & & \\
{|L_{q_1}|} \arrow[dr, "t_{L_{q_1}}"] & & & & {|L_{q_2}|} \arrow[llll, "\iota_f^L(q_1)"'] \arrow[dr, "k^L"] \arrow[drrr, "t_{L_{q_2}}"] & & & \\
 & TG_1 & & & & X_{f_T} \arrow[llll, "f_T^L"] \arrow[rr, "f_T^R"'] & & TG_2
\end{tikzcd}$$

The grammar morphisms in [6] rely on the assumption of having a fixed choice of pullbacks. Consequently the pullback-retyping construction is deterministic and morphisms are required to preserve the initial graphs and the productions "on the nose". This requirement is very strict and it may imply the absence of a morphism between two grammars having isomorphic initial graph and productions. The notion of morphism just introduced is, in a sense, more liberal: we avoid a global choice of pullbacks, and, influenced by the notion of graph process in [1], we fix "locally", for each morphism f, only part of the pullback diagrams, namely the morphisms in the family ι_f.

It is worth noticing that, for technical convenience, the partial mapping on production names is represented as a total mapping by enriching the target set with a distinguished point $\emptyset$, representing "undefinedness". In this way the condition asking the preservation of productions (Condition 2) faithfully rephrases the situation of net theory where the pre- and post-set of a transition on which the morphism is undefined are necessarily mapped to the empty multiset.

As in [6] one can show that grammar morphisms are "simulations" in the sense that for every derivation ρ_1 in $\mathcal{G}_1$ there is a corresponding derivation ρ_2 in $\mathcal{G}_2$, related to ρ_1 by the pullback-retyping construction induced by the morphism. As already observed, as a consequence of the partial arbitrariness in the choice of the pullback components, such correspondence, differently from [6], is not "functional".

3 Nondeterministic occurrence grammars

Nondeterministic occurrence grammars, as introduced in [3], are intended to represent the computations of graph grammars in a 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. 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. Furthermore, because of the dangling condition, also the graphical structure of the state imposes some precedences between productions.

A first step towards the definition of occurrence grammar is a suitable notion of safeness [8], generalizing the usual one for P/T nets which requires that each place contains at most one token in any reachable marking.

Definition 6 ((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 is injective.*

Without loss of generality, injective typed graphs can be identified with the corresponding subgraphs of the type graph (just thinking of injective morphisms as inclusions). In particular, each TG-typed graph G reachable in a safe grammar can be identified with the subgraph $t_G(|G|)$ of the type graph TG. 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 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. 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. Consider, for instance, the grammar $\mathcal{G}$ in Fig. 3, where the typing morphisms for the initial graph and the productions are represented by suitably labelling the involved graphs with items of the type graph TG. The pre-set, context and post-set of production q_1 are ${}^\bullet q_1 = \{A, L\}$, $\underline{q_1} = \{B\}$ and $q_1{}^\bullet = \{C\}$, while for the node B, ${}^\bullet B = \emptyset$, $\underline{B} = \{q_1\}$ and $B^\bullet = \{q_2\}$.

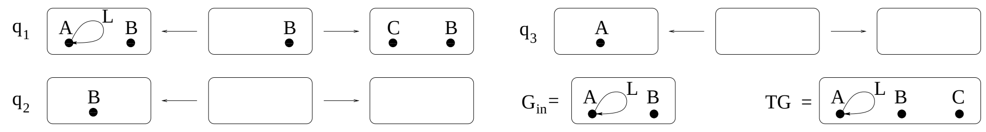

Fig. 3. The safe grammar $\mathcal{G}$.

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 7 (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 $\{q \in P : q \leq x\}$.

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 $\mathcal{G}$ of Fig. 3), 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 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 [2, 16, 13]).

Definition 8 (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 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).

Definition 9 ((nondeterministic) occurrence grammar). *A* (nondeterministic) occurrence grammar *is a 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 the asymmetric conflict $\nearrow$ is acyclic on $\lfloor q \rfloor$;*
2. *the initial graph G_{in} 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. *each item x in TG is created by at most one production in P, namely $\mid {}^\bullet x \mid \leq 1$;*
4. *for each production q, the typing t_{L_q} is injective on the "consumed part" $|L_q| - l_q(|K_q|)$, and similarly t_{R_q} is injective on the "produced part" $|R_q| - r_q(|K_q|)$.*

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

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 [2]). In particular, in Condition (1), 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 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.

It is worth stressing that because of the dangling condition, some productions of an occurrence grammar might never be applicable, as, for example, the production q_3 of grammar $\mathcal{G}$ in Fig. 3. The reason why we did not consider the dangling condition in the definition of occurrence grammar is that checking such negative (non-monotonic) condition on a production, would require to find a possible computation which removes the potentially dangling arcs, and to verify the consistency of such computation with the production at hand. By using the Turing completeness of DPO graph grammars, it can be shown that such verification is undecidable for infinite occurrence grammars, which can be obtained as unfolding of finite grammars.

The restrictions to the behaviour imposed by the dangling condition are considered when defining the configurations of an occurrence grammar, which represent exactly all the possible deterministic runs of the grammar.

Definition 10 (configuration). *A* configuration *of an occurrence graph 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 first two conditions are equivalent to those defining configurations of asymmetric event structures and thus of occurrence contextual nets [2]. Condition 3, instead, formalizes the dangling condition. If a configuration contains a production q consuming a node n and a production q' producing an arc e with source (or target) n, then arc e must be removed by some production in the configuration, otherwise, due to the dangling condition, q could not be executed. Similar considerations apply if the arc e is present in the initial graph, i.e., ${}^\bullet e = \emptyset$.

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 grammar $\mathcal{G}$ in Fig. 3, $\mathit{Conf}(\mathcal{G}) = \{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}\}$.

4 Nondeterministic graph processes

In the theory of Petri nets the notion of occurrence net is strictly related to that of process. A (non)deterministic net process is a (non)deterministic occurrence net with a suitable morphism to the original net. Similarly, nondeterministic occurrence grammars can be used to define a notion of *nondeterministic graph processes*, generalizing the deterministic graph processes of [8, 1].

A *nondeterministic graph process* is aimed at representing in a unique "branching" structure several possible computations of a grammar. The underlying occurrence grammar makes explicit the causal structure of such computations since each production can be applied at most once and each items of the type graph can be "filled" at

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

most once. Via the morphism to the original grammar, productions and items of the type graph in the occurrence grammar can be thought of, respectively, as instances of applications of productions and instances of items generated in the original grammar by such applications. Actually, to allow for such an interpretation, some further restrictions must be imposed on the process morphism. Recall that process morphisms in Petri net theory must map places into places (rather than into multisets of places) and must be total on transitions [11]. Similarly, for graph process morphisms the left component of the type-span is required to be an isomorphism in such a way that the type-span can be thought of simply as a graph morphism. Furthermore a process morphism cannot map a production to the empty production, a requirement corresponding to totality.

Definition 11 (strong morphism). *A grammar morphism $f : \mathcal{G}_1 \to \mathcal{G}_2$ is called* strong *if $f_T^L : X_f \to TG_1$ is an isomorphism and $f_P(q_1) \neq \emptyset$, for any $q_1 \in P_1$.*

Hereafter we will always choose as concrete representative of the type-span of a strong grammar morphism f, a span f_T such that the left component f_T^L is the identity id_{TG_1}.

It is not difficult to verify that, if f is a strong morphism then, by Condition 1 of the definition of grammar morphism (Definition 5), $\iota_f^{in} : |G_{in_2}| \to |G_{in_1}|$ is an isomorphism. Similarly, by Condition 2, for each production $q_1 \in P_1$, $\iota_f(q_1)$ is a triple of isomorphisms, namely each production of $\mathcal{G}_1$ is mapped to a production of $\mathcal{G}_2$ with associated isomorphic (untyped) span.

Definition 12 (graph process). *Let $\mathcal{G}$ be a graph grammar. A* graph process *of $\mathcal{G}$ is a strong grammar morphism $\chi : \mathcal{O}_\chi \to \mathcal{G}$, where $\mathcal{O}_\chi$ is an occurrence grammar.*

We will denote by TG_χ, G_{in_χ}, P_χ and π_χ the components of the occurrence grammar O_χ underlying a process χ.

Using the notions above we are naturally led to the definitions of *deterministic* occurrence grammar and process. In fact we can take an occurrence grammar $\mathcal{O}$ to be *deterministic* if the set P of its productions is a configuration of $\mathcal{O}$. Then a process χ is *deterministic* if the underlying occurrence grammar $\mathcal{O}_\chi$ is deterministic. Nicely, deterministic finite processes are exactly the (non-concatenable) graph processes of [1], which are shown there to be equivalent with the more classical trace semantics (e.g., as described in [7]).

5 Unfolding construction

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 strong grammar morphism $\chi_{\mathcal{G}}$ to the original grammar, making it a process 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 13 (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 $x, y \in G$.*

Another basic ingredient of the unfolding is the *gluing* operation. It can be seen 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.

Definition 14 (gluing). *Let q be a production, G a graph and $m : L_q \to G$ 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 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 $*$. Notice 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 $K_q \overset{r_q}{\hookrightarrow} R_q$.

As described below, the unfolding of a grammar is obtained as the limit of a chain of occurrence grammars, each approximating the unfolding up to a certain causal depth.

Definition 15 (depth). *Let $\mathcal{O} = \langle TG, P, \pi \rangle$ be an occurrence grammar. The function $depth : Elem(\mathcal{O}) \to \mathbb{N}$ is defined inductively as follows:*

$$\begin{array}{ll} depth(x) = 0 & \text{for } x \in |G_{in}| = Min(\mathcal{O}); \\ depth(q) = \max\{depth(x) \mid x \in {}^\bullet q \cup \underline{q}\} + 1 & \text{for } q \in P; \\ depth(x) = depth(q) & \text{for } x \in q^\bullet. \end{array}$$

It is not difficult to prove that *depth* is a well-defined total function, since infinite descending chains of causality are disallowed in occurrence grammars. Moreover, given an occurrence grammar $\mathcal{O}$, the grammar containing only the items of *depth* less or equal to n, denoted by $\mathcal{O}^{[n]}$, is a well-defined occurrence grammar.

As expected an occurrence grammar $\mathcal{O}$ is the (componentwise) union of its subgrammars $\mathcal{O}^{[n]}$, of depth n. Moreover it is not difficult to see that if $g : \mathcal{O} \to \mathcal{G}$ is a grammar morphism, then for any $n \in \mathbb{N}$, g restricts to a morphism $g^{[n]} : \mathcal{O}^{[n]} \to \mathcal{G}$. In particular, if $TG^{[n]}$ denotes the type graph of $\mathcal{O}^{[n]}$, then the type-span of $g^{[n]}$ will be the equivalence class of

$$\begin{tikzcd} TG^{[n]} & X^{[n]} \arrow[l, "g_T^{L[n]}"'] \arrow[r, "g_T^{R[n]}"] & TG_{\mathcal{G}} \end{tikzcd}$$

where $X^{[n]} = \{x \in X_g \mid g_T^L(x) \in TG^{[n]}\}$. Vice versa each morphism $g : \mathcal{O} \to \mathcal{G}$ is uniquely determined by its truncations at finite depths.

We are now ready to present the unfolding construction.

Definition 16 (unfolding). *Let $\mathcal{G} = \langle TG, G_{in}, P, \pi\rangle$ be a (consuming) graph grammar. We inductively define, for each n, an occurrence grammar $\mathcal{U}(\mathcal{G})^{[n]} = \langle TG^{[n]}, P^{[n]}, \pi^{[n]}\rangle$ and a morphism $\chi^{[n]} = \langle {\chi_T}^{[n]}, {\chi_P}^{[n]}, \iota^{[n]}\rangle : \mathcal{U}(\mathcal{G})^{[n]} \to \mathcal{G}$. Then the* unfolding *$\mathcal{U}(\mathcal{G})$ and the* folding morphism *$\chi_{\mathcal{G}} : \mathcal{U}(\mathcal{G}) \to \mathcal{G}$ are the occurrence grammar and strong grammar morphism defined as the componentwise union of $\mathcal{U}(\mathcal{G})^{[n]}$ and $\chi^{[n]}$, respectively.*

Since each morphism $\chi^{[n]}$ is strong, assuming that the left component of the type-span ${\chi_T}^{[n]}$ is the identity on $TG^{[n]}$ we only need to define the right component ${\chi_T^R}^{[n]} : TG^{[n]} \to TG$, which, by the way, makes $\langle TG^{[n]}, {\chi_T^R}^{[n]}\rangle$ a TG-typed graph.

$(\mathbf{n = 0})$ *The components of the grammar $\mathcal{U}(\mathcal{G})^{[0]}$ are $TG^{[0]} = |G_{in}|$, $P^{[0]} = \pi^{[0]} = \emptyset$, while morphism $\chi^{[0]} : \mathcal{U}(\mathcal{G})^{[0]} \to \mathcal{G}$ is defined by ${\chi_T^R}^{[0]} = t_{G_{in}}$, ${\chi_P}^{[0]} = \emptyset$, and ${\iota^{[0]}}^{in} = id_{|G_{in}|}$.*

$(\mathbf{n \to n+1})$ *The occurrence grammar $\mathcal{U}(\mathcal{G})^{[n+1]}$ is obtained by extending $\mathcal{U}(\mathcal{G})^{[n]}$ with all the possible production applications to quasi-concurrent subgraphs of its the type graph. More precisely, let $M^{[n]}$ be the set of pairs $\langle q, m\rangle$ such that $q \in P$ is a production in $\mathcal{G}$ and $m : L_q \to \langle TG^{[n]}, {\chi_T^R}^{[n]}\rangle$ is a match satisfying the identification condition, with $m(|L_q|)$* quasi-concurrent *subgraph of $TG^{[n]}$. Then $\mathcal{U}(\mathcal{G})^{[n+1]}$ is the occurrence grammar resulting after performing the following steps for each $\langle q, m\rangle \in M^{[n]}$.*

- *Add to $P^{[n]}$ the pair $\langle q, m\rangle$ as a new production name and extend ${\chi_P}^{[n]}$ so that ${\chi_P}^{[n]}(\langle q, m\rangle) = q$. Intuitively, $\langle q, m\rangle$ represents an occurrence of q, where the match m is needed to record the "history".*
- *Extend the type graph $TG^{[n]}$ by adding to it a copy of each item generated by the application q, marked by $\langle q, m\rangle$ (in order to keep trace of the history). The morphism ${\chi_T^R}^{[n]}$ is extended consequently. More formally, the TG-typed graph $\langle TG^{[n]}, {\chi_T^R}^{[n]}\rangle$ is replaced by $glue_{\langle q,m\rangle}(q, m, \langle TG^{[n]}, {\chi_T^R}^{[n]}\rangle)$.*
- *The production $\pi^{[n]}(\langle q, m\rangle)$ has the same untyped span of $\pi(q)$ and the morphisms $\iota^{[n]}(\langle q, m\rangle)$ are identities, that is $\iota(\langle q, m\rangle) = \langle id_{|L_q|}, id_{|K_q|}, id_{|R_q|}\rangle$. The typing of the left-hand side and of the interface is determined by m, and each item x of the right-hand side which is not in the interface is typed over the corresponding new item $\langle x, \langle q, m\rangle\rangle$ of the type graph.*

It is not difficult to verify that for each n, $\mathcal{U}(\mathcal{G})^{[n]}$ is a (finite depth) nondeterministic occurrence grammar, and $\mathcal{U}(\mathcal{G})^{[n]} \subseteq \mathcal{U}(\mathcal{G})^{[n+1]}$, componentwise. Therefore $\mathcal{U}(\mathcal{G})$ is a well-defined occurrence grammar. Similarly for each $n \in \mathbb{N}$ we have that $\chi^{[n]}$ is a well-defined morphism from $\mathcal{U}(\mathcal{G})^{[n]}$ to $\mathcal{G}$, which is the restriction to $\mathcal{U}(\mathcal{G})^{[n]}$ of $\chi^{[n+1]}$. This induces a unique morphism $\chi_{\mathcal{G}} : \mathcal{U}(\mathcal{G}) \to \mathcal{G}$.

It is possible to show that the unfolding construction applied to an occurrence grammar yields a grammar which is isomorphic to the original one.

6 Functorial unfolding for semi-weighted grammars

The unfolding construction has been defined, up to now, only at "object level". This section makes a further step towards a full correspondence with Winskel's construction, by facing the problem of characterizing the unfolding as a coreflection between the categories of graph grammars and of occurrence grammars. As in the case of (contextual) Petri nets [15, 2], we restrict to a full subcategory **SGG** of **GG** where objects satisfy conditions analogous to those defining semi-weighted P/T Petri nets. Then we show that the unfolding construction can be extended to a functor $\mathcal{U} : \mathbf{SGG} \to \mathbf{OGG}$ that is right adjoint to the inclusion functor $\mathcal{I}_O : \mathbf{OGG} \to \mathbf{SGG}$ and thus establishes a coreflection between **SGG** and **OGG**.

A graph grammar is semi-weighted if the initial graph is injective and the right-hand side of each production is injective if restricted to produced items (namely, items which are not in the interface). It is possible to show that, if we encode a Petri net N as a grammar $\mathcal{G}_N$, as sketched in Section 2, then N is a semi-weighted net if and only if $\mathcal{G}_N$ is a semi-weighted grammar.

Definition 17 (semi-weighted grammars). *A TG-typed production $L \leftarrow K \to R$ is called* semi-weighted *if t_R is injective on the "produced part" of R, namely on $|R| - r(|K|)$. A grammar $\mathcal{G}$ is called* semi-weighted *if the initial graph G_{in} is injective and for any $q \in P$ the production $\pi(q)$ is semi-weighted. We denote by* **SGG** *the full subcategory of* **GG** *having semi-weighted grammars as objects.*

The coreflection result strongly relies on the technical property which is stated in the next lemma. This is a key point where the restriction to semi-weighted grammars plays a rôle, since, as we will see, the lemma fails to hold for arbitrary grammars.

Lemma 1. *Let $\mathcal{G} = \langle TG, G_{in}, P, \pi \rangle$ be a semi-weighted grammar, let $\mathcal{O} = \langle TG', G'_{in}, P', \pi' \rangle$ be an occurrence grammar and let $f : \mathcal{O} \to \mathcal{G}$ be a grammar morphism. Then the morphism k, such that $f_T\{\iota_f^{in}, k\}(G_{in}, G_{in'})$ (see Definition 5, Condition 1) is uniquely determined. Similarly, for each $q \in P$, with $q' = f_P(q)$, the morphisms k^L, k^K and k^R such that $f_T\{\iota_f^X(q), k^X\}(X_q, X_{q'})$ for $X \in \{L, K, R\}$ (see Definition 5, Condition 2) are uniquely determined.*

A relevant property of morphisms between occurrence grammars which plays a central rôle in the proof of the coreflection is the fact that they "preserve" quasi-concurrency. This lemma can be proved along the same lines of an analogous result which hold for morphisms of contextual nets [2]. In fact, the notion of quasi-concurrency disregards the dangling condition, taking into account only causality and asymmetric conflict, whose treatment is basically the same for grammars and contextual nets.

Lemma 2 (preservation of concurrency). *Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be occurrence grammars, let $f : \mathcal{O}_1 \to \mathcal{O}_2$ be a grammar morphism, and consider, for $i \in \{1, 2\}$, a TG_i-typed graph G_i. If $f_T(G_1, G_2)$ and $t_{G_1}(|G_1|)$ is a quasi-concurrent subgraph of TG_1 then $t_{G_2}(|G_2|)$ is a quasi-concurrent subgraph of TG_2.*

Occurrence grammars are particular semi-weighted grammars, thus we have the inclusion functor $\mathcal{I}_O : \mathbf{OGG} \to \mathbf{SGG}$. The next theorem shows that the unfolding of a grammar $\mathcal{U}(\mathcal{G})$ and the folding morphism $\chi_{\mathcal{G}}$ are cofree over $\mathcal{G}$. Therefore $\mathcal{U}$ extends to a functor that is right adjoint of $\mathcal{I}_O$, thus establishing a coreflection between **SGG** and **OGG**.

Theorem 1 (coreflection between SGG and OGG). *Let $\mathcal{G}$ be a semi-weighted grammar, let $\mathcal{U}(\mathcal{G})$ be its unfolding and let $\chi : \mathcal{U}(\mathcal{G}) \to \mathcal{G}$ be the folding morphism as in Definition 16. Then for any occurrence grammar $\mathcal{O}$ and morphism $g : \mathcal{O} \to \mathcal{G}$ there exists a unique morphism $h : \mathcal{O} \to \mathcal{U}(\mathcal{G})$ such that the following diagram commutes:*

$$\begin{tikzcd}
\mathcal{U}(\mathcal{G}) \arrow[r, "\chi"] & \mathcal{G} \\
\mathcal{O} \arrow[u, dotted, "h"] \arrow[ur, "g"'] &
\end{tikzcd}$$

Therefore $\mathcal{I}_O \dashv \mathcal{U}$.

The proof of the existence of the morphism h uses Lemma 2 to inductively define, for each n, a morphism $h^{[n]} : \mathcal{O}^{[n]} \to \mathcal{U}(\mathcal{G})$, while uniqueness basically relies on Lemma 1.

7 Conclusions and future work

A natural question regards the possibility of extending the result of this paper to the whole category **GG** of graph grammars. We remark that the proof of the uniqueness of the morphism h in Theorem 1 strongly relies on Lemma 1 which in turn requires the grammar $\mathcal{G}$ to be semi-weighted. Unfortunately the problem does not reside in our proof technique: the cofreeness of the unfolding $\mathcal{U}(\mathcal{G})$ and of the folding morphism $\chi_{\mathcal{G}}$ over $\mathcal{G}$ may really fail to hold if the grammar $\mathcal{G}$ is not semi-weighted.

For instance, consider grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ in Fig. 4, where typed graphs are represented by decorating their items with pairs "concrete identity:type". The grammar $\mathcal{G}_2$ is not semi-weighted since the initial graph is not injective, while $\mathcal{G}_1$ is clearly an occurrence grammar. The unfolding $\mathcal{U}(\mathcal{G}_2)$ of the grammar $\mathcal{G}_2$, according to Definition 16, is defined as follows. The initial graph and type graph of $\mathcal{U}(\mathcal{G}_2)$ coincide with $|G_{in_2}|$. Furthermore, $\mathcal{U}(\mathcal{G}_2)$ contains two productions $q_2' = \langle q_2, m' \rangle$ and $q_2'' = \langle q_2, m'' \rangle$, which are two occurrences of q_2 corresponding to the two possible different matches $m', m'' : L_{q_2} \to G_{in_2}$ (the identity and the swap).

Now, let $g : \mathcal{G}_1 \to \mathcal{G}_2$ be a grammar morphism, with $g_P(q_1) = q_2$ and the type span g_T defined as follows: X_{g_T} is a discrete graph with two nodes x and y, $g_T^L(x) = g_T^R(y) = A$ and $g_T^L(x) = g_T^R(y) = B$ (see the bottom row of the diagram in Fig. 4). Consider the pullback-retyping diagram in Fig. 4, expressing the preservation of the initial graph for morphism g (Condition 1 of Definition 5). Notice that there are two possible different morphisms k and k' from $|G_{in_2}|$ to X_{g_T} (represented via plain and dotted arrows, respectively) making the diagram commutes and the square a pullback. This provides a counterexample, showing that Lemma 1 cannot be extended to general (non semi-weighted) grammars.

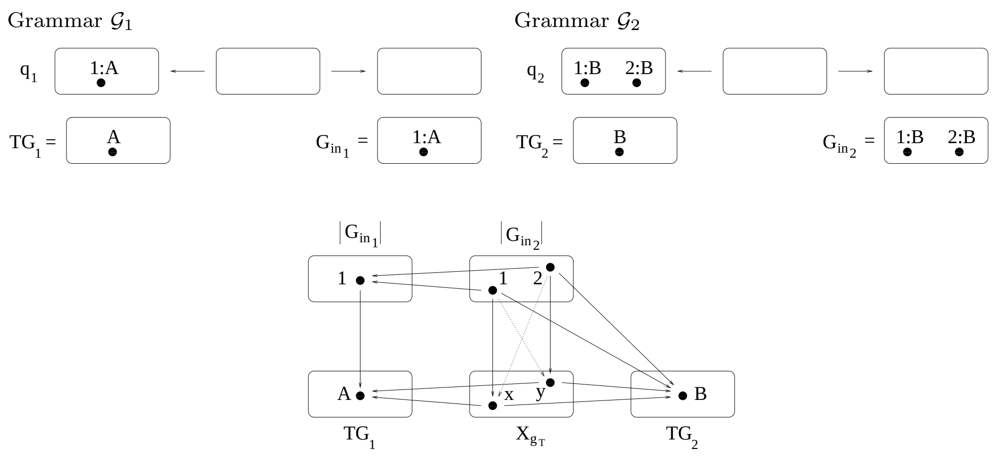

Fig. 4. The grammars $\mathcal{G}_1$ and $\mathcal{G}_2$, and the pullback-retyping diagram for their initial graphs.

Now, it is not difficult to see that, correspondingly, we can construct two different morphisms $h_i : \mathcal{G}_1 \to \mathcal{U}(\mathcal{G}_2)$ $(i \in \{1, 2\})$, such that $\chi_{\mathcal{G}_2} \circ h_i = g$, the first one mapping production q_1 into q_2' and the second one mapping q_1 into q_2''. An immediate consequence of this fact is the impossibility of extending $\mathcal{U}$ to morphisms, in order to obtain a functor which is right adjoint of the inclusion $\mathcal{I} : \mathbf{OGG} \to \mathbf{GG}$.

A possible way to overcome this problem could be the choice of a different notion of grammar morphism, constraining in some way also the "k"-component of the pullback-retyping diagram. Some insights could come again from the theory of Petri nets [15], where the treatment of general P/T nets reveals similar problems which are solved there via the notions of decorated occurrence net and family morphism, at the price of obtaining a proper adjunction rather than a coreflection.

To conclude, it is worth stressing that a similar construction has been proposed by Ribeiro in her doctoral thesis [17] for the *single-pushout (SPO) approach*. She defines an unfolding functor from the category of graph grammars to a category of (abstract) *occurrence grammars*, showing that it is a right adjoint to a suitable *folding functor*. Although the basic ideas are very similar, concretely, the differences between the two settings, like the absence of the application conditions in the SPO approach, a different notion of "enabling" allowing for the concurrent application of productions related by asymmetric conflict and a different choice of grammar morphisms, makes difficult a synthetic direct comparison. For lack of space we defer to the full version a detailed analysis of the relation between the two approaches.

References

1. P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 283–295. Springer Verlag, 1998.

2. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378, pages 63–80. Springer Verlag, 1998.
3. P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578, pages 73–89. Springer Verlag, 1999.
4. R. Banach and A. Corradini. An Opfibration Account of Typed DPO and DPB Graph Transformation: General Productions. Technical Report UMCS-96-11-2, University of Manchester, Department of Computer Science, 1996.
5. A. Corradini. Concurrent Graph and Term Graph Rewriting. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR'96*, volume 1119 of *LNCS*, pages 438–464. Springer Verlag, 1996.
6. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and J. Padberg. The category of Typed Graph Grammars and its adjunctions with categories of derivations. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proceedings of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *LNCS*. Springer Verlag, 1996.
7. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An Event Structure Semantics for Graph Grammars with Parallel Productions. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proceedings of the 5th International Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *LNCS*. Springer Verlag, 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 G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations.* World Scientific, 1997.
10. H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 3–14. Springer Verlag, 1987.
11. U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
12. 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.
13. R. Langerak. *Transformation and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.
14. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In *Proceedings CONCUR '92*, volume 630 of *LNCS*, pages 286–301. Springer Verlag, 1992.
15. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Place/Transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1997.
16. G. M. Pinna and A. Poigné. On the nature of events: another perspective in concurrency. *Theoretical Computer Science*, 138:425–454, 1995.
17. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* PhD thesis, Technische Universität Berlin, 1996.
18. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer Verlag, 1987.