On the concurrent semantics of Algebraic Graph Grammars*

Paolo Baldan[1] and Andrea Corradini[2]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
baldan@dsi.unive.it andrea@di.unipi.it

Abstract. Graph grammars are a powerful model of concurrent and distributed systems which can be seen as a proper extension of Petri nets. Inspired by this correspondence, a truly concurrent semantics has been developed along the years for the algebraic approaches to graph grammars, based on Winskel's style unfolding constructions as well as on suitable notions of processes. A basic role is played in this framework by the study of contextual and inhibitor nets, two extensions of ordinary nets which can be seen as intermediate models between ordinary Petri nets and algebraic graph grammars.
This paper presents a survey of these results, discussing in a precise, even if informal way, some of the main technical contributions that made possible the development of such a theory.

Introduction

Petri nets [40,42] are one of the most widely used models of concurrency. Since their introduction they have attracted the interest of both theoreticians and practitioners. Along the years Petri nets have been equipped with satisfactory semantics, doing justice to their intrinsically concurrent nature. These semantics have served as basis for the development of a variety of modelling and verification techniques. However, the simplicity of Petri nets, which is one of the reasons of their success, represents also a limit in their expressiveness. If one is interested in giving a more structured description of the state, or if the kind of dependencies between steps of computation cannot be reduced simply to causality and conflict, Petri nets are likely to be inadequate.

This paper summarizes the work presented by the authors in a series of papers [2,3,4,7,8,9,10,12], most of which written jointly with Ugo Montanari, and some with Nadia Busi, Michele Pinna and Leila Ribeiro. Such papers are the outcome of a project aimed at proposing graph transformation systems as an alternative model of concurrency, extending Petri nets. The basic intuition underlying the use of graph transformation systems for formal specifications is to represent the states of a system as graphs (possibly attributed with data-values) and state

* Research partially supported by the EU FET-GC Project IST-2001-32747 AGILE, and by the EC RTN 2-2001-00346 SEGRAVIS.

transformations by means of rule-based graph transformations. Needless to say, the idea of representing system states by means of graphs is pervasive in computer science. Whenever one is interested in giving an explicit representation of the interconnections, or more generally of the relationships among the various components of a system, a natural solution is to use (possibly hierarchical and attributed) graphs. The possibility of giving a suggestive pictorial representation of graphical states makes them adequate for the description of the meaning of a system specification, even to a non-technical audience. A popular example of graph-based specification language is given by the Unified Modelling Language (UML), but we recall also the more classical Entity/Relationship (ER) approach, or Statecharts, a specification language suited for reactive systems. Moreover, graphs provide a privileged representation of systems consisting of a set of processes communicating through ports.

When one is interested in modelling the *dynamic aspects* of systems whose states have a graphical nature, graph transformation systems are clearly one of the most natural choices. Since a graph rewriting rule has only a local effect on the state, it is natural to allow for the parallel application of rules acting on independent parts of the state, so that a notion of concurrent computation naturally emerges in this context. The research in the field, mainly that dealing with the so-called *algebraic approaches* to graph transformation [25,22,27], has led to the attempt of equipping graph grammars with a satisfactory semantical framework, where their truly concurrent behaviour can be suitably described and analyzed. After the seminal work [31], which introduced the notion of *shift equivalence*, many original contributions to the theory of concurrency for algebraic graph transformation systems have been proposed during the last ten years, most of them inspired by their relation with Petri nets. In particular, for the *double-pushout* (DPO) approach to graph transformation, building on some ideas of [31], a *trace semantics* has been proposed in [18,22]. Resorting to a construction in the style of Mazurkiewicz, the trace semantics has been used to derive an *event structure semantics* [20,19] for DPO graph grammars. Graph grammars have been endowed also with a process semantics with the introduction of *graph processes* [21], further refined with the notion of concatenable (deterministic) processes [8]. A Winskel's style unfolding construction [51] has been defined both for the *single pushout* (SPO) and the DPO approaches [43,9,10], and has been exploited for providing, through suitable chains of functors, such grammars with more abstract semantics based on event structures and domains.

In this survey paper, after recalling the basics of the algebraic approaches to graph transformation and their relationship with Petri nets, we will summarize the functorial, unfolding semantics of Petri nets and the elegant way in which it can be reconciled with the event structure semantics based on deterministic processes. Next we will discuss how this approach has been generalized to algebraic graph grammars. This required the definition of new structures and constructions that will be briefly outlined in the following sections. We shall focus mainly on the definition of two generalizations of prime event structures, called *asymmetric* and *inhibitor* event structures, explaining why they were necessary, and to which extent they made possible to generalize to graph grammars the constructions and results originally developed for Petri nets. It is worth stressing here that this research activity contributed to the theory of Petri nets as well, by generalizing the functorial semantics to the classes of *contextual* and *inhibitor nets*, already introduced in the literature. Such nets can be considered as intermediate models between Place/Transition (P/T) nets and graph grammars.

The rest of this paper is organized as follows. In Section 1 we introduce the DPO and SPO approaches to graph transformation, discussing their relation with Petri nets, and we stress the role of contextual and inhibitor nets as intermediate models between Petri nets and graph grammars. This allows us to organize the mentioned models in an ideal partial order where each model generalizes its predecessors. Then Section 2 outlines the approach to the truly concurrent semantics of ordinary Petri nets which is proposed as a paradigm. Section 3 describes the semantical framework that has been generalized from Petri nets to graph grammars, and Section 4 gives an overview of the results, by explaining how and to what extent such semantical framework has been lifted along the chains of models, first to contextual and inhibitor nets and then to DPO and SPO graph grammars. Finally, Section 5 discusses some open problems and directions of future research.

1 Graph grammars and their relation with Petri nets

In this section we present the algebraic approaches to graph transformation and we discuss how ordinary Petri nets can be seen as special algebraic graph grammars. The new features with which graph grammars extend ordinary Petri nets establish a close relationship between graph grammars and two generalizations of Petri nets in the literature, i.e., contextual and inhibitor nets.

1.1 The algebraic approaches to graph transformation

Generally speaking, a graph grammar consists of a start graph together with a set of *graph productions*, i.e., rules of the kind $p : L \rightsquigarrow R$, specifying that, under certain conditions, once an occurrence (a *match*) of the left-hand side L in a graph G has been detected, it can be replaced by the right-hand side R. The form of graph productions, the notion of match, and the mechanisms establishing how a production can be applied to a graph, and what the resulting graph is, depend on the specific graph rewriting formalism.

Here we consider the *algebraic approaches* to graph rewriting [25,18,27], where the basic notions of production and direct derivation are defined in terms of constructions and diagrams in a suitable category. Consequently, the resulting theory is very general and flexible, easily adaptable to a very wide range of structures, simply by changing the underlying category.

In the *double-pushout* approach, a graph production consists of a left-hand side graph L, a right-hand side graph R and a (common) interface graph K embedded both in R and in L, as depicted in the top part of Fig. 1. Informally,

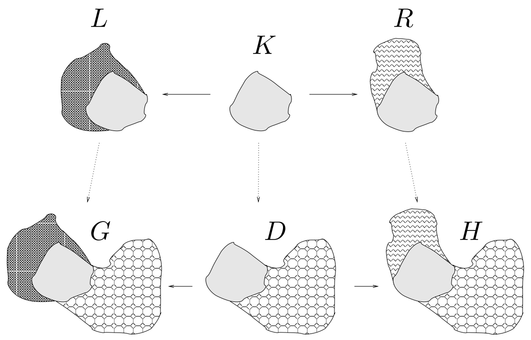

Fig. 1. A (double-pushout) graph rewriting step.

to apply such a rule to a graph G we must find a *match*, namely an occurrence of its left-hand side L in G. The rewriting mechanism first removes the part of the left-hand side L which is not in the interface K producing the graph D, and then adds the part of the right-hand side R which is not in the interface K, thus obtaining the graph H. Formally, this is obtained by requiring the two squares in Fig. 1 to be pushouts in the category of graphs and *total* graph morphisms, hence the name of the approach. The interface graph K is "preserved": it is necessary to perform the rewriting step, but it is not affected by the step itself. Notice that the interface K plays a fundamental role in specifying how the right-hand side has to be glued with the graph D. Working with productions having an empty interface graph K, the expressive power would drastically decrease: only disconnected subgraphs could be added.

In the *single-pushout* approach, a production consists instead of a partial, injective graph morphism $p : L \rightarrowtail R$ from the left- to the right-hand side graph. By looking at the partial morphism p as a span of total morphisms

$$L \hookleftarrow dom(p) \rightarrow R$$

one sees that the domain of p plays here the role of the interface K of DPO rules. To apply such a production to a given match of L in a graph G, i.e., to a *total* morphism $L \rightarrow G$, we have to compute the pushout of $p : L \rightarrowtail R$ and $L \rightarrow G$ in the category of graphs and *partial* graph morphisms.

The most relevant difference between the DPO and SPO approaches (see [27]) is the fact that while the construction of the double pushout diagram may fail if the match $L \rightarrow G$ does not satisfy the so-called *gluing conditions* with respect to the given rule, the construction of the pushout of an SPO rule $L \rightarrowtail R$ and a match $L \rightarrow R$ is always possible. We shall come back on this when relevant in the rest of the paper.

1.2 Relation with Petri nets

A basic observation belonging to the folklore (see, e.g., [17] and references therein) regards the close relationship existing between graph grammars and Petri nets. Basically a Petri net can be viewed as a graph transformation system that acts on a restricted kind of graphs, namely discrete, labelled graphs (that can be considered as sets of tokens labelled by places), the productions being the transitions of the net. For instance, Fig. 2 presents a Petri net transition t and the corresponding DPO and SPO graph productions which consume nodes corresponding to two tokens in s_0 and one token in s_1 and produce new nodes corresponding to one token in s_2 and one token in s_3. The interface is empty in the DPO rule and the domain of the morphism is empty in the SPO rule, since nothing is explicitly preserved by a net transition. It is easy to check that both representations satisfy the properties one would expect: the production can be applied to a given marking if and only if the corresponding transition is enabled, and the double or single pushout construction produces the same marking as the firing of the transition.

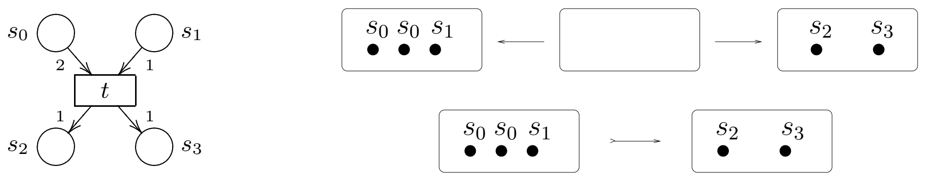

Fig. 2. A Petri net transition and the corresponding DPO and SPO productions.

In this view, general graph transformation systems can be seen as a *proper* extension of ordinary Petri nets in two dimensions:

1. they allow for general productions, possibly with non-empty interface, specifying rewriting steps where a part of the state is *preserved*, i.e., required, but not affected by the rewriting step;
2. they allow for a *more structured description of the state*, that is an arbitrary, possibly non-discrete, graph.

The first capability is essential to give a faithful representation of concurrent accesses to shared resources. In fact, the part of the state preserved in a rewriting step, i.e., the (image of the) interface graph in the DPO or the domain of the production in the SPO approach, can be naturally interpreted as a part of the state which is accessed in a read-only manner by the rewriting step. Coherently with such interpretation, several productions can be applied in parallel sharing (part of) the interface. It is worth remarking that the naïve technique of representing a read operation as a consume/produce cycle may cause a loss of concurrency since it imposes an undesired serialization of the read-only accesses to the shared resource.

As for the second capability, even if multisets may be sufficient in many situations, as already mentioned in the introduction, graphs are more appropriate when one is interested in giving an explicit representation of the interconnections among the various components of the systems, e.g., if one wants to describe the topology of a distributed system and the way it evolves.

These distinctive features of graph grammars establish a link with two extensions of ordinary Petri nets in the literature, introduced to overcome some deficiencies of the basic model: *contextual nets* and *inhibitor nets*.

1.3 Contextual nets

Contextual nets [37], also called nets with test arcs in [16], activator arcs in [30] or read arcs in [49], extend ordinary nets with the possibility of checking for the presence of tokens which are not consumed. Concretely, besides the usual preconditions and postconditions, a transition of a contextual net has also some *context* conditions, which specify that the presence of some tokens in certain places is necessary to enable the transition, but such tokens are not affected by the firing of the transition. Following [37], non-directed (usually horizontal) arcs are used to represent context conditions: for instance, transition t in the left part of Fig. 3 has place s as context.

Clearly the context of a transition in a contextual nets closely corresponds to the interface graph of a DPO production and to the domain of an SPO production, seen as a partial morphism. As suggested by Fig. 3, a contextual net corresponds to a graph grammar still acting on discrete graphs, but where productions may have a non-empty interface/domain.

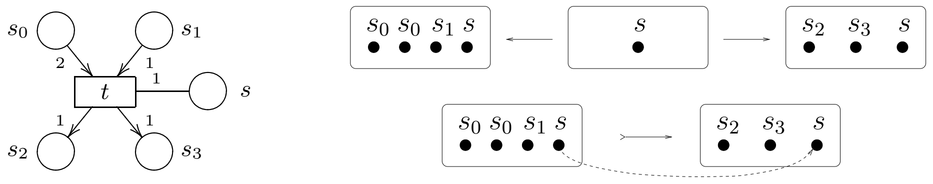

Fig. 3. A contextual Petri net transition and the corresponding DPO and SPO productions.

For their ability of faithfully representing concurrent read-only accesses to shared resources, contextual nets have been used to model the concurrent access to shared data (e.g., for serializability problems for concurrent transactions in a database) [23,44], to give a concurrent semantics to concurrent constraint programs [14] where several agents access a common store, to model priorities [29] and to compare temporal efficiency in asynchronous systems [49].

1.4 Inhibitor nets

Inhibitor nets (or nets with *inhibitor arcs*) [1] further generalize contextual nets with the possibility of checking not only for the presence, but also for the *absence* of tokens in a place. For each transition an *inhibitor set* is defined and the transition is enabled only if no token is present in the places of its inhibitor set. When a place s is in the inhibitor set of a transition t we say that s *inhibits* (the firing of) t. The fact that a place s inhibits a transition t is graphically represented by drawing a dotted line from s to t, ending with an empty circle, as shown in the left part of Fig. 4.

While, at a first glance, this could seem a minor extension, it definitely increases the expressive power of the model. In fact, many other extensions of ordinary nets, like nets with reset arcs or prioritized nets, can be simulated in a direct way by using nets with inhibitor arcs (see, e.g., [39]). Indeed the crucial observation is that ordinary nets can easily simulate all the operations of RAM machines, with the exception of the *zero-testing*. Enriching nets with inhibitor arcs is the simplest extension which allows to overcome this limit, thus giving the model the computational power of Turing machines.

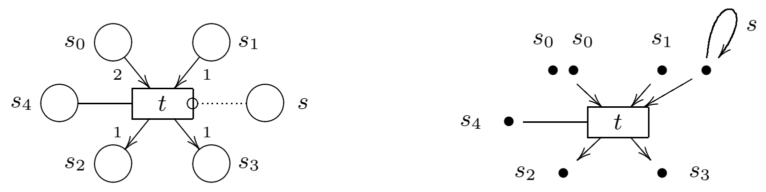

Fig. 4. Correspondence between inhibitor Petri nets and DPO graph grammars.

In this case the relation with algebraic graph grammars is less straightforward, and it only concerns the DPO approach. We must recall that in a graph transformation system each rewriting step is required to preserve the consistency of the graphical structure of the state, namely each step must produce a well-defined graph. Hence, as required by a part of the application condition of the DPO approach, the so-called *dangling condition*, a production q which removes a node n cannot be applied if there are edges having n as source or target, which are not removed by q: in fact, such edges would remain *dangling* in the resulting graph. In other words the presence of such edges *inhibits* the application of q. This is informally illustrated by Fig. 4, where place s which inhibits transition t in the left part, becomes an edge which would remain dangling after the execution of t, in the right part. As in the case of contextual nets, this intuitive relation can be made formal, but here, for lack of space, we cannot give the details of the correspondence.

It is worth stressing, again informally, that in the SPO approach the dangling condition is not necessary. By the nature of pushouts in the category of graphs and partial morphisms, a rule which deletes a node can be applied to any match of its left-hand side: any edge attached to that node is automatically erased by the construction, as a kind of side-effect.

2 Truly concurrent semantics of Petri nets

Along the years Petri nets have been equipped with several semantics, aimed at describing, at the right degree of abstraction, the truly concurrent nature of their computations. The approach that we propose as a paradigm, comprises the semantics based on *deterministic processes*, whose origin dates back to an early proposal by Petri himself [41] and the semantics based on the *nondeterministic unfolding*, introduced in a seminal paper by Nielsen, Plotkin and Winskel [38], and shows how the two may be reconciled in a satisfactory framework.

2.1 Deterministic process semantics

The notion of *deterministic process* naturally arises when trying to give a truly concurrent description of net computations, taking explicitly into account the *causal* dependencies ruling the occurrences of events in *single* computations.

The prototypical example of Petri net process is given by the *Goltz-Reisig processes* [28]. A Goltz-Reisig process of a net N is a (deterministic) occurrence net O, i.e., a finite net enjoying suitable acyclicity and conflict freeness properties, plus a mapping to the original net $\varphi : O \to N$. The flow relation induces a partial order on the elements of the net O, which can be naturally interpreted as causality. The mapping essentially labels places and transitions of O with places and transitions of N, in such a way that places in O can be thought of as tokens in a computation of N and transitions of O as occurrences of transition firings in such computation. For instance, Fig. 5 depicts a Petri net and a deterministic process of such a net, representing the sequential execution of two occurrences of t_1 followed by t_2, in parallel with t_3.

A refinement of Goltz-Reisig processes, the so-called *concatenable processes* [24], form the arrows of a category $\mathbf{CP}[N]$, where objects are markings (states of the net) and arrow composition models the sequential composition of computations. It turns out that such category is a *symmetric monoidal category*, in which the tensor product represents faithfully the parallel composition of processes.

2.2 Unfolding semantics

A deterministic process represents only a single, deterministic computation of a net. Nondeterminism is captured implicitly by the existence of several different "non confluent" processes having the same source. An alternative classical approach to the semantics of Petri nets is based on an *unfolding construction*, which maps each net into a single branching structure, representing all the possible events that can occur in all the possible computations of the net and the

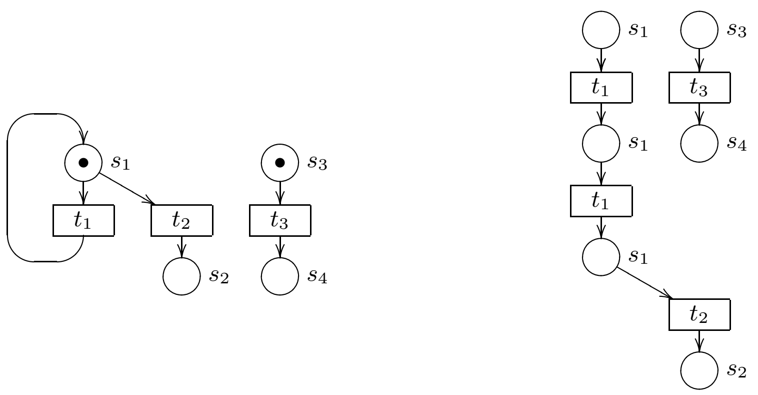

Fig. 5. A Petri net and a deterministic process for the net.

relations existing between them. This structure expresses not only the causal ordering between the events, but also gives an explicit representation of the branching (choice) points of the computations.

In the seminal work of Nielsen, Plotkin and Winskel [38], the denotation of a *safe net* is a *coherent finitary prime algebraic Scott domain* [47] (briefly *domain*), obtained via a construction which first unfolds the net into a (nondeterministic) occurrence net which is then abstracted to a prime event structure, which, finally, gives rise to a domain. Building on such result, Winskel [51] proves the existence of a chain of categorical coreflections (a particularly nice kind of adjunction), leading from the category **S-N** of safe (marked) P/T nets to the category **Dom** of finitary prime algebraic domains, through the categories **O-N** of occurrence nets and **PES** of prime event structures.

$$\begin{tikzcd}
\mathbf{S\text{-}N} \arrow[r, shift right, "\mathcal{U}"'] \arrow[r, phantom, "\perp"] & \mathbf{O\text{-}N} \arrow[l, hook', shift right, "\mathcal{I}_{\mathbf{Occ}}"'] \arrow[r, shift right, "\mathcal{E}"'] \arrow[r, phantom, "\perp"] & \mathbf{PES} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \mathbf{Dom} \arrow[l, shift right, "\mathcal{P}"']
\end{tikzcd}$$

The first step unwinds a safe net N into a *nondeterministic occurrence* net $\mathcal{U}(N)$, which can be seen as a "complete" nondeterministic process of the net N, representing in its branching structure *all* the possible computations of the original net N. The construction exploits the fact that in a safe Petri net, a specific occurrence of a transition t can be identified uniquely by its *history*, namely by the finite set of transition occurrences starting from the initial marking which are strictly necessary to enable to considered occurrence of t. Any two distinct transition occurrences t_1 and t_2 can be related in four possible, mutually exclusive, ways:

1. t_2 is *causally dependent* on t_1 (denoted $t_1 < t_2$) if any computation including t_2 includes also t_1;
2. t_1 is causally dependent on t_2 ($t_2 < t_1$) in the symmetric case;

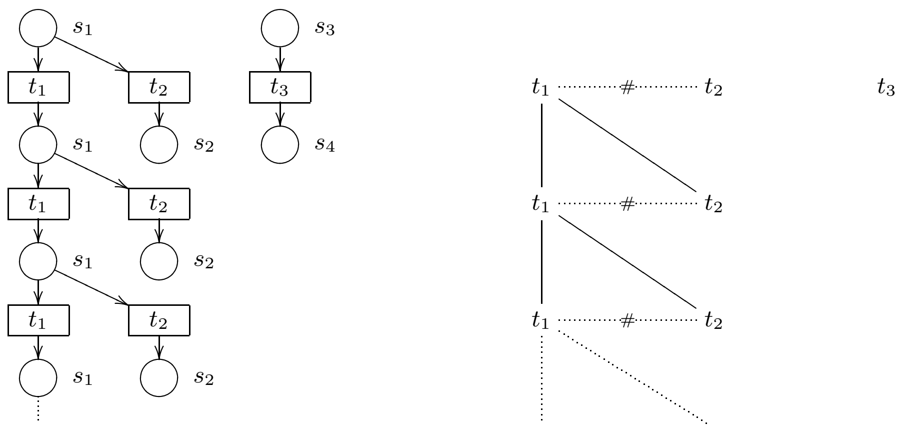

Fig. 6. Unfolding and event structure semantics of Petri nets.

3. t_1 and t_2 are in *conflict* ($t_1 \# t_2$) if they do not appear together in any computation;
4. t_1 and t_2 are *concurrent* if none of the previous conditions holds.

The relations of causality and conflict are easily shown to be generated by the *direct causality*, which relates a transition occurrence which produces a token with all those which consume it, and by the *direct conflict*, which relates two transition occurrences which would consume the same token. The occurrence net $\mathcal{U}(N)$ obtained as the unfolding of a safe net N records exactly all this information.

The subsequent step abstracts such occurrence net to a *prime event structure* (PES). The PES is obtained from the unfolding simply by forgetting the places, and remembering only the transition occurrences and the causality and conflict relations among them. From a prime event structure E it is possible to generate freely an occurrence net $\mathcal{N}(E)$ which is the "most general" among those having E as underlying PES. Such a net is obtained by considering the events of E as transition occurrences, and introducing, among others, one fresh place for every pair of events related by causality or conflict in E, in order to enforce the same relationships in $\mathcal{N}(E)$.

The last step (which establishes an equivalence between the category of prime event structures and the category of domains) maps any event structure to its domain of configurations. Fig. 6 presents the unfolding and event structure corresponding to the net in Fig. 5.

In [36] it has been shown that essentially the same construction applies to the category of *semi-weighted* nets, i.e., P/T nets in which the initial marking is a set and transitions can generate at most one token in each post-condition. Besides strictly including safe nets, semi-weighted nets also offer the advantage of being characterized by a "static condition", not involving the behaviour but just the structure of the net.

2.3 Reconciling deterministic processes and unfolding

Since the unfolding is essentially a "maximal" nondeterministic process of a net, one would expect the existence of a clear relation between the unfolding and the deterministic process semantics. Indeed, as shown in [35], the domain associated to a net N through the unfolding construction can be equivalently characterized as the set of deterministic processes of the net starting from the initial marking, endowed with a kind of prefix ordering. This result is stated in an elegant categorical way by resorting to concatenable processes. Given a (semi-weighted) net N with initial marking m, the comma category $\langle m \downarrow \mathbf{CP}[N] \rangle$ is shown to be a preorder, whose elements are intuitively finite computations starting from the initial state, and if φ_1 and φ_2 are elements of the preorder, $\varphi_1 \preceq \varphi_2$ when φ_1 can evolve to φ_2 by performing appropriate steps of computation. Then the ideal completion of such preorder, which includes also the infinite computations of the net, is shown to be isomorphic to the domain generated from the unfolding.

$$\begin{tikzcd}
 & \textit{Deterministic processes} \arrow[dr, bend left] & \\
\textit{P/T Nets} \arrow[ur, bend left] \arrow[dr, bend right] & & \textit{Domains} \\
 & \textit{Unfolding} \arrow[ur, bend right] &
\end{tikzcd}$$

3 Concurrent semantics: from nets to graph grammars

In this section, guided by the relationship between graph grammars and Petri nets, we describe the way the semantical framework described in the previous section has been generalized to graph grammars.

The main complications which arise in the treatment of graph grammars are related to the possibility of expressing rewritings where part of the state is preserved and, just for the DPO approach, to the need of preserving the consistency of the graphical structure of the state, a constraint which leads to the mentioned "inhibiting effects" between production applications. Therefore, not surprisingly, contextual and inhibitor nets play an essential role in the extension in that they offer a technically simple framework, where problems which are conceptually relevant to graph grammars can be studied in isolation.

Intuitively, we can organize the considered formalisms in an ideal partial ordering leading from Petri nets to graph transformation systems

$$\begin{tikzcd}
\text{Petri nets} \arrow[r, dotted] & \text{Contextual nets} \arrow[r, dotted] \arrow[dr, dotted] & \text{Inhibitor nets} \arrow[r, dotted] & \text{DPO graph grammars} \\
 & & \text{SPO graph grammars} &
\end{tikzcd}$$

and for each one of such formalisms we develop a similar theory by following a common schema which can be summarized as follows:

1. We define a *category of systems* **Sys**, where morphisms, which basically origin from an algebraic view of the systems, can be interpreted as simulations.
2. We develop an *unfolding semantics*, expressed as a coreflection between **Sys** and a subcategory **O-Sys**, where objects, called "occurrence" systems, are suitable systems exhibiting an acyclic behaviour. From the unfolding we extract an (appropriate kind of) *event structure*, the transformation being expressed as a functor from **O-Sys** to the considered category of event structures **ES**. In the case of contextual nets and of SPO grammars this functor establishes a coreflection between **O-Sys** and **ES**. Finally, a connection is established with domains and PES by showing that the category **ES** of generalized event structures coreflects into the category **Dom** of domains.
   Summing up, we obtain the following chain of functors, leading from systems to event structures and domains
   $$\begin{tikzcd} \mathbf{Sys} \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathbf{O\text{-}Sys} \arrow[l, hook', shift right] \arrow[r] & \mathbf{ES} \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathbf{Dom} \arrow[l, shift right] \arrow[r, shift right] \arrow[r, phantom, "\sim"] & \mathbf{PES} \arrow[l, shift right] \end{tikzcd}$$
   The last step in the chain is the equivalence between the categories **Dom** of domains and **PES** of prime event structures, due to Winskel.
3. We introduce a notion of *deterministic process* for systems in **Sys**. Relying on the work in point (2), a general (possibly nondeterministic) *process* of a system $\mathcal{S}$ is defined as an "occurrence system" in **O-Sys**, plus a (suitable kind) of morphism back to the original system $\mathcal{S}$ (the prototypical example of nondeterministic process being the unfolding). Then, roughly speaking, a process is *deterministic* if it contains no conflict, or, in other words, if the corresponding event structure has a configuration including all the events.
   The deterministic processes of a system $\mathcal{S}$ are turned into a category $\mathbf{CP}[\mathcal{S}]$, by endowing them with a notion of concatenation, modelling the sequential composition of computations.
4. We show that the deterministic process and the unfolding semantics can be reconciled by proving that, as for ordinary nets, the comma category $\langle \mathit{Initial\ State} \downarrow \mathbf{CP}[\mathcal{S}]\rangle$, is a preorder whose ideal completion is isomorphic to the domain obtained from the unfolding, as defined at point (2).

It is fair to point here that the steps (3) and (4) above have not been completely worked out for SPO grammars.

Observe that, differently from what happens for ordinary nets, the unfolding semantics (essentially based on nondeterministic processes) is defined before developing a theory of deterministic processes. To understand why, note that for ordinary nets the only source of nondeterminism is the the presence of pairs of different transitions with a common precondition, and therefore there is an obvious notion of "deterministic net". When considering contextual nets, inhibitor nets or graph grammars the situation becomes less clear: the dependencies between event occurrences cannot be described only in terms of causality and conflict, and the deterministic systems cannot be given a purely syntactical

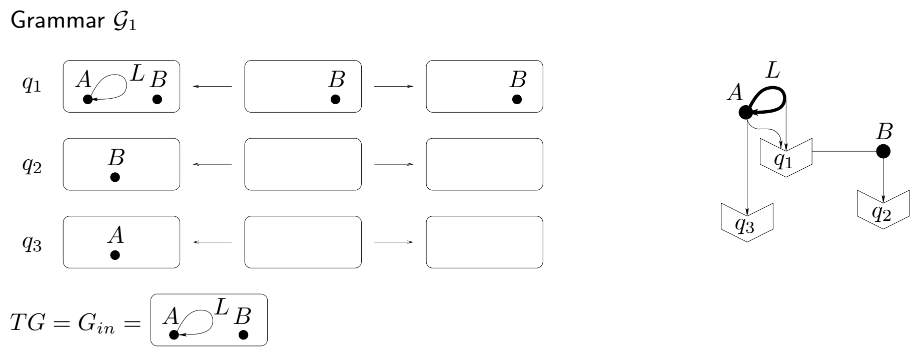

Fig. 7. The safe graph grammar $\mathcal{G}_1$ and its net-like representation.

characterization. Consequently, a clear understanding of the structure of non-deterministic computations becomes essential to be able to single out which are the good representatives of deterministic computations.

4 Some insights into the technical problems

For each one of the considered models the core of the developed theory is point (2) and, more specifically, the formalization of the kind of dependencies among events which can occur in their computations. As mentioned above, such dependencies cannot be faithfully reduced to causality and conflict and thus appropriate generalizations of Winskel's event structures must be defined. Next we give some more details on the specific problems that we found for each formalism and on the way we decided to face them.

4.1 From prime to asymmetric event structures

In the case of algebraic graph grammars, both in the DPO and in the SPO approaches, the presence of a context in a production, i.e., of items that are needed for the application of a production but which are not consumed, introduces a new kind of dependency among production occurrences, making prime event structures not completely satisfactory as a semantic domain.

As an example, consider the (typed) DPO graph grammar $\mathcal{G}_1$ of Figure 7. On the left-hand side, the grammar is represented as usual in the DPO approach, consisting of a set of productions (spans of injective graph morphisms) and a start graph, all of them typed over the type graph TG (i.e., equipped with a homomorphism to TG), which, in this case, coincides with the start graph G_{in}. On the right-hand side, a net-like pictorial representation of the same grammar is shown, where the productions and the items of the type graph play the rôle of transitions and of places of a Petri net, respectively. This net-like representation can be given for *strongly safe* graph grammars, which, intuitively, are the graph grammar counterpart of safe nets (see [10] for more details).

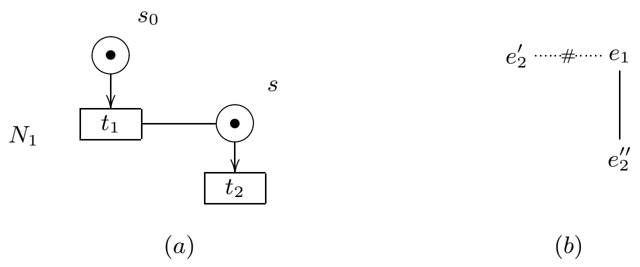

Fig. 8. A simple contextual net and a prime event structure representing its behaviour.

Let us focus on productions q_1 and q_2. Both are applicable to the start graph G_{in}. But notice that if we apply q_2 first, then q_1 cannot be applied anymore because q_2 deletes node B; on the other hand, if we apply q_1 first, then q_2 can still be applied. This phenomenon has been extensively studied for *contextual nets*. For example, in the net N_1 of Fig. 8(a), transitions t_1 and t_2 play the same rôle as productions q_1 and q_2 of the above grammar, and place s, which is a context of q_1 and a precondition of q_2, is like node B above.

The possible firing sequences in net N_1 are given by the firing of t_1, the firing of t_2, and the firing of t_1 followed by t_2, denoted $t_1;t_2$, while $t_2;t_1$ is not allowed. This situation cannot be modelled in a direct way within a prime event structure: t_1 and t_2 are neither in conflict nor concurrent nor causally dependent. Simply, as for an ordinary conflict, the firing of t_2 prevents t_1 to be executed, so that t_1 can never follow t_2 in a computation, but the converse is not true, since t_2 *can* fire after t_1. This situation can be interpreted naturally as an *asymmetric conflict* between the two transitions. Equivalently, since t_1 precedes t_2 in any computation where both transitions fire, t_1 acts as a cause of t_2 in such computations. However, differently from a true cause, t_1 is not necessary for t_2 to be fired. Therefore we can also think of the relation between the two transitions as a *weak* form of *causality.*

A possible way to encode this situation in a PES is to represent the firing of t_1 with an event e_1 and the firing of t_2 with two distinct mutually exclusive events: e'_2, representing the execution of t_2 that prevents t_1, thus in conflict with e_1, and e''_2, representing the execution of t_2 after t_1, thus caused by e_1. Such PES is depicted in Fig. 8.(b), where causality is represented by a plain arrow and conflict is represented by a dotted line, labelled by $\#$. However, this solution is not completely satisfactory with respect to the interpretation of contexts as "read-only resources": since t_1 just reads the token in s without changing it, one would expect the firing of t_2, preceded or not by t_1, to be represented by a single event.

In order to provide a more direct, event based representation of contextual net computations, *asymmetric event structure* (AES) were introduced in [7]. An AES, besides of the usual causality relation $\leq$ of a prime event structure, has a relation $\nearrow$, called the *asymmetric conflict relation*, that allows one to specify

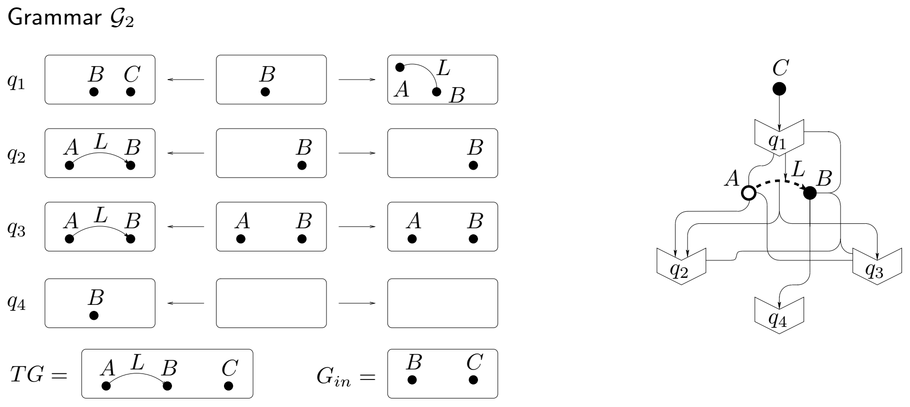

Fig. 9. Graph grammar $\mathcal{G}_2$ and its net-like representation.

the new kind of dependency described above simply as $t_1 \nearrow t_2$. Informally, in an AES each event has a set of "strong" causes (given by the causality relation) and a set of weak causes (due to the presence of the asymmetric conflict relation). To be fired, each event must be preceded by all strong causes and by a (suitable) subset of the weak causes. Therefore, differently from PES's, an event of an AES can have more than one history. Moreover the usual symmetric binary conflict $e\#e'$ can be represented easily by using cycles of asymmetric conflicts: if $e \nearrow e'$ and $e' \nearrow e$ then clearly e and e' can never occur in the same computation, since each one should precede the other.

The main result of [7] shows that Winskel's functorial semantics for safe nets can be generalized to the following, similar chain of adjuctions for contextual nets, where asymmetric event structures play a central rôle.

$$\begin{tikzcd}
\textbf{Semi-weighted Contextual Nets} \arrow[r, phantom, "\perp"] \arrow[r, bend right, "\mathcal{U}_a"'] & \textbf{Occurrence Contextual Nets} \arrow[l, hook'] \arrow[r, "\mathcal{E}_a"'] \arrow[r, phantom, "\perp"] & \textbf{Asymmetric Event Structures} \arrow[l, "\mathcal{N}_a"'] \arrow[r, "\mathcal{L}_a"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, "\mathcal{P}_a"']
\end{tikzcd}$$

4.2 From asymmetric to inhibitor event structures

Unfortunately, AES's are not yet sufficient to capture all relationships among the production occurrences in a DPO graph grammar. Consider grammar $\mathcal{G}_2$ of Figure 9 (in the net-like representation, on the right-hand side, nodes with empty interior and dashed edges can be seen as empty places). More specifically, let us focus on the relationships among the various productions. Notice that q_4 can be applied to the start graph G_{in} consisting of nodes B and C, but if we first apply q_1, then the application of q_4 is prevented by the dangling condition: removing the node B would leave edge L without its target node, so, basically, q_4 cannot be applied for ensuring a structural property of the state. Production q_4 could be applied again if we first delete edge L, by applying production q_2 or q_3.

Such complex relationships have been analyzed in depth for *inhibitor Petri nets*. Consider the inhibitor net N_2 in Fig. 10 where the place s, which inhibits transition t, is in the post-set of transition t' and in the pre-set of t_0. The execution of t' inhibits the firing of t, which can be enabled again by the firing of t_0. Thus t can fire before or after the "sequence" $t'; t_0$, but not in between the two transitions. Roughly speaking there is a sort of atomicity of the sequence $t'; t_0$ with respect to t. The situation can be more involved since many transitions t_0,

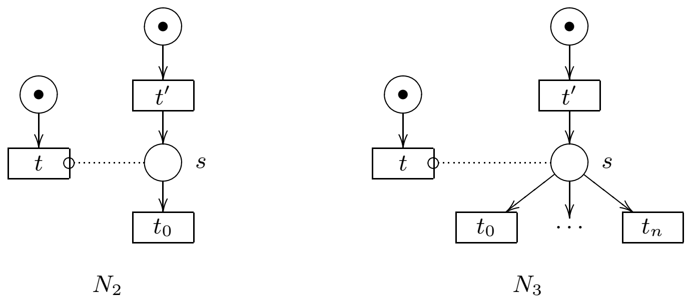

Fig. 10. Two inhibitor nets.

$\dots$, t_n may have the place s in their pre-set (see the net N_3 in Fig. 10). Therefore, after the firing of t', the transition t can be re-enabled by any of the conflicting transitions $t_0, \dots, t_n$. This leads to a sort of *or*-causality, but only when t fires after t'. With a logical terminology we can say that t causally depends on the implication $t' \Rightarrow t_0 \vee t_1 \vee \dots \vee t_n$.

In order to model these complex relationships in a direct way, a generalization of PES's and AES's has been introduced, called *inhibitor event structures* (IES's). A IES is equipped with a ternary relation, called *DE-relation (disabling-enabling relation)* and denoted by $\vdash\!\circ\,(\cdot,\cdot,\cdot)$, which allows one to model the dependencies between transitions in N_3 as $\vdash\!\circ\,(\{t'\}, t, \{t_0, \dots, t_n\})$. It is possible to show that the DE-relation is sufficient to represent both causality and asymmetric conflict and thus, concretely, it is the only relation of an IES.

Using inhibitor event structures and the DE-relation as basic tools, a functorial semantics in Winskel's style has been proposed for (semi-weighted) inhibitor nets in [4,3] as summarised by the following diagram:

$$\begin{tikzcd}
\textbf{Semi-weighted Inhibitor Nets} \arrow[r, shift right, "\mathcal{U}_i"'] \arrow[r, phantom, "\perp"] & \textbf{Occurrence Inhibitor Nets} \arrow[l, hook', shift right] \arrow[r, "\mathcal{E}_i"'] & \textbf{Inhibitor Event Structures} \arrow[r, shift right, "\mathcal{L}_i"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_i"']
\end{tikzcd}$$

Besides of the fact that inibitor event structures replace asymmetric ones, even at this level of abstraction, it is possible to see another relevant difference between the functorial semantics of semi-weighted inhibitor nets and the simpler case of contextual nets. In fact, the functor from the category of inhibitor occurrence nets to the category of IES's *does not have* a left adjoint and thus the whole semantic transformation is not expressed as a coreflection. Indeed, by making only very mild assumptions, it has been shown in [3] that such a left adjoint does not exist, essentially because of the presence of a restricted kind of or-causality in inhibitor occurrence nets.

4.3 Lifting the results to DPO graph grammars

When we finally turn our attention to DPO graph grammars we are rewarded of the effort spent on generalized Petri nets, since basically nothing new has to be invented. Inhibitor event structures are expressive enough to model the structure of DPO graph grammar computations and the theory developed for inhibitor nets smoothly lifts, at the price of some technical complications, to DPO grammars. Furthermore, not only the process and the unfolding semantics developed for DPO graph grammars are shown to agree, but also they have been shown to be consistent with the classical theory of concurrency for DPO grammar in the literature, basically relying on *shift-equivalence*. More specifically:

1. A Winskel's style semantics for DPO graph grammars is presented in [2,9,10], as summarized by the following diagram:

$$\begin{tikzcd}
\textbf{DPO Graph Grammars} \arrow[r, shift right, "\mathcal{U}_g"'] \arrow[r, phantom, "\perp"] & \textbf{DPO Occurrence Grammars} \arrow[l, hook', shift right] \arrow[r, "\mathcal{E}_g"'] & \textbf{Inhibitor Event Structures} \arrow[r, shift right, "\mathcal{L}_i"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_i"']
\end{tikzcd}$$

   The unfolding construction associates to each graph grammar a nondeterministic occurrence grammar describing its behaviour. Such a construction establishes a coreflection between suitable categories of DPO grammars and the category of occurrence grammars. The unfolding is then abstracted to an inhibitor event structure and finally to a prime algebraic domain (or equivalently to a prime event structure).
2. *Nondeterministic graph processes* are introduced in [2,10], generalizing the deterministic processes of [21]. The notion fits nicely in the theory since a graph process of a DPO grammar $\mathcal{G}$ is defined simply as a (special kind of) grammar morphism from an occurrence grammar to $\mathcal{G}$, while in [21] an *ad hoc* mapping was used.
3. *Concatenable graph processes* are introduced in [8], as a variation of deterministic, finite processes, endowed with an operation of concatenation which models sequential composition of computations. The appropriateness of this notion is confirmed by the fact that the category $\mathbf{CP}[\mathcal{G}]$ of concatenable processes of a DPO grammar $\mathcal{G}$ turns out to be isomorphic to the truly concurrent model of computation of $\mathcal{G}$, as defined in [22] using the classical notions of shift-equivalence and of traces.
4. The event structure obtained via the unfolding is shown in [9] to coincide both with the one defined in [20] via a comma category construction on the category of concatenable derivation traces, and with the one proposed in [46], based on a deterministic set-theoretical variant of the DPO approach. These results, besides confirming the appropriateness of the proposed unfolding construction, give an unified view of the various event structure semantics for the DPO approach to graph transformation.

4.4 Unfolding semantics of SPO graph grammars

Recently, a Winskel's style unfolding semantics has been developed for the SPO approach to graph transformation as well, as reported in [12]. Apart from the technical differences in the way rules are defined, the main difference with respect to the DPO approach lies in the fact that there are no conditions on rule applications, i.e., whenever a match is found the corresponding rule can always be applied.

It turned out that a coreflective unfolding semantics for SPO graph grammars can be defined, leading to the following chain of adjunctions:

$$\begin{tikzcd}
\textbf{SPO Graph Grammars} \arrow[r, shift right, "\mathcal{U}_s"'] \arrow[r, phantom, "\perp"] & \textbf{SPO Occurrence Grammars} \arrow[l, hook', shift right] \arrow[r, shift right, "\mathcal{E}_s"'] \arrow[r, phantom, "\perp"] & \textbf{Asymmetric Event Structures} \arrow[l, shift right, "\mathcal{N}"'] \arrow[r, shift right, "\mathcal{L}_a"'] \arrow[r, phantom, "\perp"] & \textbf{Domains} \arrow[l, shift right, "\mathcal{P}_a"']
\end{tikzcd}$$

The first step of the above diagram is obtained as a slight variation of the unfolding construction for SPO grammars proposed in [43]. The rest of the construction differs from and improves that for DPO graph grammars recalled above, for the following facts:

– Due to the absence of the dangling condition, asymmetric event structures are sufficient to represent adequately the dependencies among production occurrences: inhibitor event structures are not necessary.
– A novel construction, inspired by the work on contextual nets [7], allows us to build a canonical occurrence SPO graph grammar $\mathcal{N}(A)$ from any given asymmetric event structure A. This provides the left-adjoint functor (indeed a coreflection) which is missing in the corresponding chain for inhibitor nets and DPO grammars. Given an asymmetric event structure A, the corresponding grammar has the events of A as productions. The graph items are freely generated in order to induce the right kind of dependencies between events. More specifically, first the nodes of the graph are freely generated according to the dependencies in A. Then for any pair of nodes, edges connecting the two nodes are freely generated according to the dependencies in A and the specific restrictions of the SPO rewriting mechanism.

5 Conclusions

In this paper we surveyed several results proposed by our coauthors and ourselves in a series of papers, contributing to the development of a systematic theory of concurrency for algebraic graph grammars, aimed at closing the existing gap between graph transformation systems and Petri nets. A second achievement of this research activity is the development of an analogous unifying theory for two widely diffused generalizations of Petri nets, namely contextual and inhibitor nets. In fact, while a theory of deterministic processes for these kind of nets was already available in the literature (see, e.g., [37,15]), the Winskel-style semantics, comprising the unfolding construction, its abstraction to a prime algebraic domain semantics, as well as its relation with the deterministic process semantics were missing.

The truly concurrent semantics for graph grammars (and generalized nets) is intended to represent the basis for defining more abstract observational semantics to be used for the analysis and verification of the modelled systems. For instance, the notions of process and of event structure associated to a process naturally lead to the definition of a behavioural equivalence, called *history preserving bisimulation* (HP-bisimulation) [48], which, differently from ordinary bisimulation, takes into account the properties of concurrency of the system. A generalization of this approach to graph grammars has been proposed in [11].

The unfolding semantics of Petri nets has been used successfully for the analysis of finite-state systems: as shown in [34], a finite fragment can be extracted from the (possibly infinite) unfolding, which is still useful to study some relevant properties of the system, like reachability, deadlock freeness, and liveness and concurrency of transitions. Such an approach has been extended to contextual nets in [50]. Inspired by this line of research, recently we started to develop, in joint works with Barbara and Bernhard König, a methodology for the verification of algebraic graph grammars using finite approximations of the unfolding, and a suitable graph logic for expressing relevant properties. Papers [5,13] address the verification of possibly infinite-state systems, while [6] is more closely related to [34] as it consider finite-state systems only.

Finally, although we considered only graph rewriting acting on directed (typed) graphs, it would be interesting to understand if the presented constructions and results can be extended to more general structures. While the generalization to hypergraphs looks trivial, developing a similar theory for more general structures and for abstract categories (e.g., High Level Replacement Systems [26], or the recently introduced Adhesive Categories [32]) is not immediate and represents an interesting topic of further investigation.

References

1. T. Agerwala and M. Flynn. Comments on capabilities, limitations and "correctness" of Petri nets. *Computer Architecture News*, 4(2):81–86, 1973.
2. 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.
3. P. Baldan, N. Busi, A. Corradini, and G.M. Pinna. Domain and event structure semantics for Petri nets with read and inhibitor arcs. *Theoretical Computer Science*, to appear, 2004.
4. P. Baldan, N. Busi, A. Corradini, and G.M. Pinna. Functorial concurrent semantics for Petri nets with read and inhibitor arcs. In C. Palamidessi, editor, *CONCUR'00 Conference Proceedings*, volume 1877 of *LNCS*, pages 442–457. Springer Verlag, 2000.
5. P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR 2001*, pages 381–395. Springer, 2001. LNCS 2154.
6. P. Baldan, A. Corradini, and B. König. Veryfing Finite-State Graph Grammars: an Unfolding-Based Approach. In *Proc. of CONCUR 2004*, pages 83–98. Springer, 2004. LNCS 3170.

7. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.
8. 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.
9. 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 of *LNCS*, pages 73–89. Springer Verlag, 1999.
10. P. Baldan, A. Corradini, and U. Montanari. Unfolding of double-pushout graph grammars is a coreflection. In G. Ehrig, G. Engels, H.J. Kreowsky, and G. Rozemberg, editors, *TAGT'98 Conference Proceedings*, volume 1764 of *LNCS*, pages 145–163. Springer Verlag, 1999.
11. P. Baldan, A. Corradini, and U. Montanari. Bisimulation Equivalences for Graph Grammars. In*Formal and Natural Computing*, W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa eds., number 2300 in LNCS, pages 158–190. Springer Verlag, 2002.
12. P. Baldan, A. Corradini, U. Montanari, and L. Ribeiro. Coreflective Concurrent Semantics for Single-Pushout Graph Grammars. *Proceedings WADT 2002*, volume 2755 of *LNCS*, pages 165–184. Springer Verlag, 2002.
13. P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proceedings of the First International Conference on Graph Transformation (ICGT 2002)*, volume 2505 of *LNCS*, pages 14–30. Springer, 2002.
14. F. Bueno, M. Hermenegildo, U. Montanari, and F. Rossi. Partial order and contextual net semantics for atomic and locally atomic CC programs. *Science of Computer Programming*, 30:51–82, 1998.
15. N. Busi. *Petri Nets with Inhibitor and Read Arcs: Semantics, Analysis and Application to Process Calculi.* PhD thesis, University of Siena, Department of Computer Science, 1998.
16. S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In M. Ajmone-Marsan, editor, *Applications and Theory of Petri Nets*, volume 691 of *LNCS*, pages 186–205. Springer Verlag, 1993.
17. A. Corradini. Concurrent graph and term graph rewriting. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 438–464. Springer Verlag, 1996.
18. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Abstract graph derivations in the double-pushout approach. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 86–103. Springer Verlag, 1994.
19. A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. An event structure semantics for safe graph grammars. In E.-R. Olderog, editor, *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pages 423–444. North-Holland, 1994.
20. 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.
21. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

22. 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 [45], chapter 3.
23. N. De Francesco, U. Montanari, and G. Ristori. Modeling Concurrent Accesses to Shared Data via Petri Nets. In *Programming Concepts, Methods and Calculi, IFIP Transactions A-56*, pages 403–422. North Holland, 1994.
24. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Informatica*, 33:641–647, 1996.
25. 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.
26. 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.
27. 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 [45], chapter 4.
28. U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
29. R. Janicki and M. Koutny. Invariant semantics of nets with inhibitor arcs. In *Proceedings of CONCUR '91*, volume 527 of *LNCS*. Springer Verlag, 1991.
30. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.
31. H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, Technische Universität Berlin, 1977.
32. S. Lack and P. Sobociński. Adhesive categories. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.
33. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.
34. K.L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.
35. J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for Place/Transition Petri nets. *Theoretical Computer Science*, 153(1-2):171–210, 1996.
36. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Place/Transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1997.
37. U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6), 1995.
38. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
39. J.L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.
40. C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Institutes für Instrumentelle Matematik, Bonn, 1962.
41. C.A. Petri. Non-sequential processes. Technical Report GMD-ISF-77-5, Gesellshaft für Mathematik und Datenverarbeitung, Bonn, 1977.
42. W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
43. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* PhD thesis, Technische Universität Berlin, 1996.
44. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets.* PhD thesis, Department of Computer Science - University of Pisa, 1994.

45. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*. World Scientific, 1997.
46. G. Schied. On relating rewriting systems and graph grammars to event structures. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 326–340. Springer Verlag, 1994.
47. D. S. Scott. Outline of a mathematical theory of computation. In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176, 1970.
48. R. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In A. Kreczmar and G. Mirkowska, editors, *Proceedings of MFCS'89*, volume 39 of *LNCS*, pages 237–248. Springer Verlag, 1989.
49. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 538–548. Springer Verlag, 1997.
50. W. Vogler, A. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 501–516. Springer-Verlag, 1998.
51. 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.