To appear in: Electronic Notes in Theoretical Computer Science 51 (2011)
URL: http://www.elsevier.nl/locate/entcs/volume51.html

Concurrency for Graph Grammars in a Petri net shell

Paolo Baldan[1,2]

Dipartimento di Informatica
Università di Pisa

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 we develop truly concurrent semantics for DPO graph grammars based on (deterministic) processes and on a Winskel's style unfolding construction, and we show that the two approaches can be reconciled. A basic role is played by the study of contextual and inhibitor nets, two extensions of ordinary nets which can be seen as intermediate models between graph grammars and ordinary Petri nets.

Keywords: Graph grammars, Petri nets, concurrent semantics, processes, unfolding, event structures, domains.

Introduction

Petri nets [35,37] are one of the the most widely used models of concurrency and they have attracted, since their introduction, the interest of both theoreticians and practitioners. Along the years Petri nets have been equipped with satisfactory semantics, making justice of their intrinsically concurrent nature, which 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 in the PhD thesis [2], which is part 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

[1] Research partially supported by the EC TMR Network GETGRATS, by the ESPRIT Working Group APPLIGRAPH, and by the MURST project TOSCA.

[2] Email: baldan@di.unipi.it

transformation systems for formal specifications is to represent the states of a system as graphs (possibly attributed with data-values) and state 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 Modeling 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 [20], has been 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 [27], which introduces the notion of *shift-equivalence*, during the last years many original contributions to the theory of concurrency for algebraic graph transformation systems have been proposed, most of them inspired by their relation with Petri nets. In particular, for the DPO approach to graph transformation, building on some ideas of [27], a *trace semantics* has been proposed in [13,17]. Resorting to a construction in the the style Mauzurkiewicz, the trace semantics has been used to derive an *event structure semantics* [15,14] for DPO graph grammars. Some steps have been moved also in the direction of providing graph grammars with a process semantics with the introduction of *graph processes* [16].

In this paper, with the aim of consolidating the foundations of the concurrency theory of graph transformation systems, we explore the possibility of extending to graph grammars some fundamental approaches to the semantics coming from the world of Petri nets. More specifically, we provide graph transformation systems with truly concurrent semantics based on concatenable (deterministic) processes [23,19] and on a Winskel's style unfolding construction [45], as well as with more abstract semantics based on event structures and domains, and we show that the various approaches can be "reconciled". As an intermediate step, we study two generalizations of Petri nets proposed in the literature, which reveal a close relationship with graph transformation systems, namely contextual nets (also called nets with read, activator or test arcs) and nets with inhibitor arcs. Due to their relatively wide diffusion, we believe that the work on these extended kinds of nets may be understood as an additional outcome, independently from its usefulness in carrying out our program on graph transformation systems.

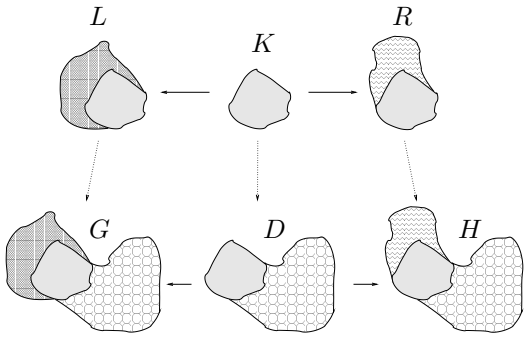


Figure 1. A (double-pushout) graph rewriting step.

The rest of this paper is organized as follows. In Section 1 we introduce the DPO approach to graph transformation, discussing its 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 chain where each model generalizes its predecessor. Then Section 2 outlines the approach to the truly concurrent semantics of ordinary Petri nets which we propose as a paradigm. Section 3 gives an overview of the results, by explaining how the semantical framework of ordinary nets can be lifted along the chain of models, first to contextual and inhibitor nets and then to graph grammars. Finally, Section 4 discusses some open problems and directions of future research.

1 Graph grammars and their relation with Petri nets

In this section we present the DPO approach to graph transformation and we discuss how ordinary Petri nets can be seen as special DPO 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 double-pushout approach to graph rewriting

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 in general the mechanisms stating how a production can be applied to a graph and what the resulting graph is, depend on the specific graph rewriting formalism.

Here we follow the so-called *double-pushout* (DPO) *algebraic approach* [20], 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 DPO 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

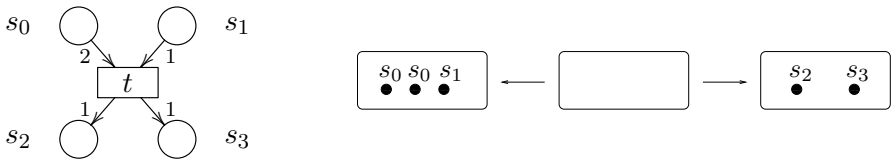


Figure 2. A Petri net transition and a corresponding DPO production.

and in L, as depicted in the top part of Fig. 1. Informally, 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, 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.

1.2 Relation with Petri nets

A basic observation belonging to the folklore (see, e.g., [12] 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 graph production which consumes nodes corresponding to two tokens in s_0 and one token in s_1 and produces new nodes corresponding to one token in s_2 and one token in s_3. The interface is empty since nothing is explicitly preserved by a net transition. 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 the double pushout construction produces the same marking as the firing of the transition.

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

(i) 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;

(ii) 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, can be naturally interpreted as a part of the state which is accessed in a read-only manner by the rewriting step. Coherently

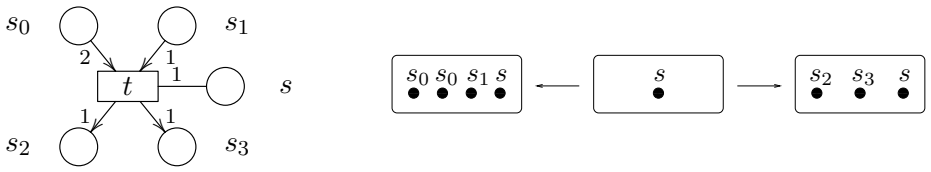


Figure 3. A contextual Petri net transition and a corresponding DPO production.

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*.

Contextual nets

Contextual nets [32], also called nets with test arcs in [11], activator arcs in [26] or read arcs in [43], 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 [32], 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. 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.

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) [18,38], to give a concurrent semantics to concurrent constraint programs [9] where several agents access a common store, to model priorities [25] and to compare temporal efficiency in asynchronous systems [43].

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

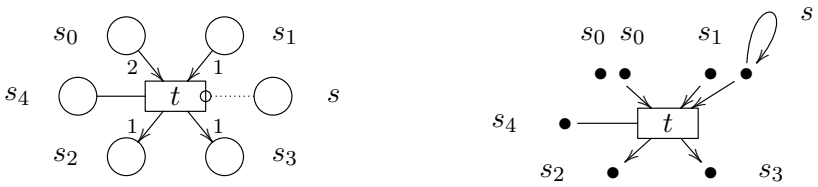


Figure 4. Correspondence between inhibitor Petri nets and DPO graph grammars.

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., [34]). Indeed the crucial observation is that traditional nets can easily simulate all the operation 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.

In this case the relation with graph grammars is less straightforward. 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, the so-called *dangling condition*, a production q which removes a node n cannot be applied if there are edges with source or target in n, not removed by q. 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 arc 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.

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 [36] and the semantics based on the *nondeterministic unfolding* introduced in a seminal paper by Nielsen, Plotkin and Winskel [33], and shows how the two may be reconciled in a satisfactory framework.

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.

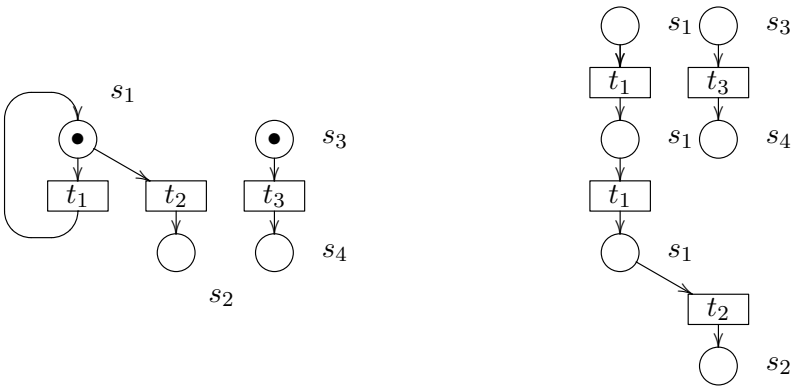


Figure 5. A Petri net and a deterministic process for the net.

The prototypical example of Petri net process is given by the *Goltz-Reisig processes* [23]. A Goltz-Reisig process of a net N is a (deterministic) occurrence net O, i.e., a finite net satisfying 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 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* [19], 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.

Unfolding semantics

A deterministic process specifies only the meaning of 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 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 [33], the denotation of a *safe net* is a *coherent finitary prime algebraic Scott domain* [41] (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. Building on such result, Winskel [45] proves the existence of a chain of categorical coreflections (a particularly nice kind of adjunction), leading from the category **S-N** of safe (marked)

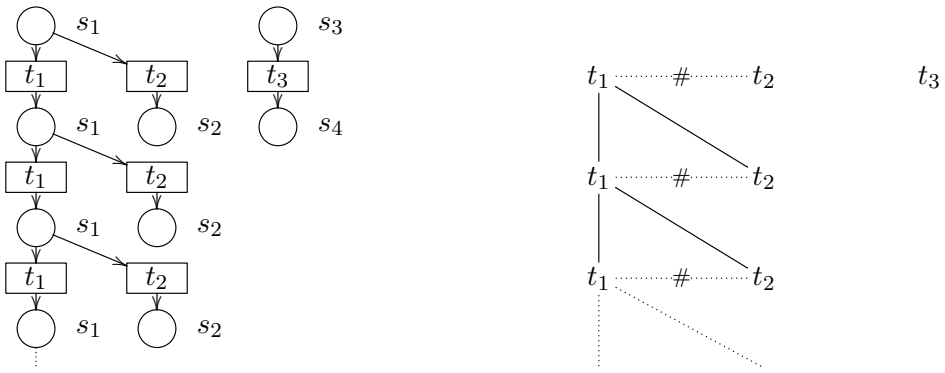


Figure 6. Unfolding and event structure semantics of Petri nets.

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.

$$\mathbf{S\text{-}N} \underset{\mathcal{U}}{\overset{\mathcal{I}_{\mathbf{Occ}} \supset}{\underset{\perp}{\leftarrow}}} \mathbf{O\text{-}N} \underset{\mathcal{E}}{\overset{\mathcal{N}}{\underset{\perp}{\leftarrow}}} \mathbf{PES} \underset{\mathcal{L}}{\overset{\mathcal{P}}{\underset{\sim}{\leftarrow}}} \mathbf{Dom}$$

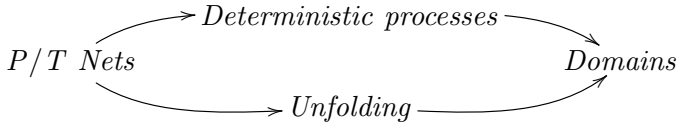
The first step unwinds a safe net N into a *nondeterministic occurrence* net $\mathcal{U}(N)$, which can be seen as a "complete" nondeterministic process of the net N, representing in its branching structure *all* the possible computations of the original net N. The subsequent step abstracts such occurrence net to a *prime event structure* (PES). The PES is obtained from the unfolding simply by forgetting the places, but remembering the basic dependency relations between transitions that they induce, namely *causality*, modelled by a partial order relation $\leq$ and *conflict*, modelled by a symmetric and irreflexive relation $\#$, hereditary with respect to causality. The last step (which establishes an equivalence between the category of prime event structures and the category of domains) maps the event structure to its domain of configurations. Fig. 6 presents the unfolding and event structure corresponding to the net in Fig. 5.

In [31] 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.

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 [30], 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.



3 Concurrent semantics: from nets to graph grammars

In this section, guided by the relationship between graph grammars and Petri nets, we explore the possibility of generalizing to graph grammars the semantical framework described in the previous section for Petri nets.

3.1 The general approach

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

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

$$\text{Petri nets} \dashrightarrow \text{Contextual nets} \dashrightarrow \text{Inhibitor nets} \dashrightarrow \text{Graph grammars}$$

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

(i) We define a *category of systems* **Sys**, where morphisms, which basically origin from an algebraic view of the systems, can be interpreted as simulations.

(ii) 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 this functor establishes a coreflection between **O-Sys** and **ES**). Finally, a connection is established with domains and traditional 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

$$\mathbf{Sys} \underset{\perp}{\overset{\longleftarrow}{\longrightarrow}} \mathbf{O\text{-}Sys} \longrightarrow \mathbf{ES} \underset{\perp}{\overset{\longleftarrow}{\longrightarrow}} \mathbf{Dom} \underset{\sim}{\overset{\longleftarrow}{\longrightarrow}} \mathbf{PES}$$

The last step in the chain is the equivalence between the categories **Dom** of domains and **PES** of prime event structures, due to Winskel.

(iii) We introduce a notion of *deterministic process* for systems in **Sys**. Relying on the work in point (ii), 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.

(iv) We show that the deterministic process and the unfolding semantics can be reconciled by proving that, as for traditional 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 (ii).

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

3.2 Some insights on the technical problems

For each one of the considered models the core of the developed theory is point (ii) 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 solve them.

Contextual nets and asymmetric conflicts

When dealing with contextual nets, the crucial point is the fact that the presence of context conditions leads to *asymmetric conflicts* or *weak dependencies* between events. Consider, for instance, the net in Fig. 7, where the same place s is a context for transition t_0 and a precondition for transition t_1. The possible firing sequences are the firing of t_0, the firing of t_1 and the firing of t_0 followed by t_1, denoted $t_0; t_1$, while $t_1; t_0$ is not allowed. This represents a new situation not arising within

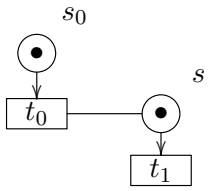


Figure 7. Asymmetric conflict in contextual nets.

ordinary net theory: t_0 and t_1 are neither in conflict nor concurrent nor causal dependent. Simply, as for a traditional conflict, the firing of t_1 prevents t_0 to be executed, so that t_0 can never follow t_1 in a computation. But the converse is not true, since t_1 *can* fire after t_0. This situation can be interpreted naturally as an *asymmetric conflict* between the two transitions. Equivalently, since t_0 precedes t_1 in any computation where both transitions are executed, in such computations t_0 acts as a cause of t_1. However, differently from a true cause, t_0 is not necessary for t_1 to be fired. Therefore we can also think of the relation between the two transitions as a *weak* form of *causal dependency*.

Prime event structures and in general Winskel's event structures result inadequate to give a faithful representation of situations of asymmetric conflict. To overcome this limitation we introduce *asymmetric event structures* (AES's), a generalization of PES's where symmetric conflict is replaced by a relation $\nearrow$ modelling *asymmetric conflict*. An AES allows us to specify the new kind of dependency described above for transitions t_0 and t_1 simply as $t_0 \nearrow t_1$.

The notion of asymmetric conflict plays an essential role both in the ordering of the configurations of an AES, which is different from set-inclusion, and in the definition of (deterministic) occurrence contextual nets, which are introduced as the net-theoretical counterpart of (deterministic) AES's. Then the entire Winskel's construction naturally lifts to contextual nets [4].

Inhibitor nets and the disabling-enabling relation

When considering inhibitor nets, the nonmonotonic features related to the presence of inhibitor arcs (negative conditions) make the situation far more complicated. First if a place s is in the post-set of a transition t', in the inhibitor set of t and in the pre-set of t_0 (see the net Fig. 8.(a)), then 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 $t_0, \ldots, t_n$ may have the place s in their pre-set (see the net in Fig. 8.(b)). Therefore, after the firing of t', transition t can be re-enabled by any of the conflicting transitions $t_0, \ldots, t_n$. This leads to a sort of *or*-causality. With a logical terminology we can say that t causally depends on the implication $t' \Rightarrow t_0 \vee t_1 \vee \ldots \vee t_n$.

To face these additional complications we introduce *inhibitor event structures* (IES's), which enrich asymmetric event structures with a ternary relation, called *DE-relation (disabling-enabling relation)*, denoted by $\mapsto(\cdot,\cdot,\cdot)$, which, for instance, models the previously described situation as $\mapsto(\{t'\}, t, \{t_0, \ldots, t_n\})$. The *DE-relation* is sufficient to represent both causality and asymmetric conflict and thus, concretely,

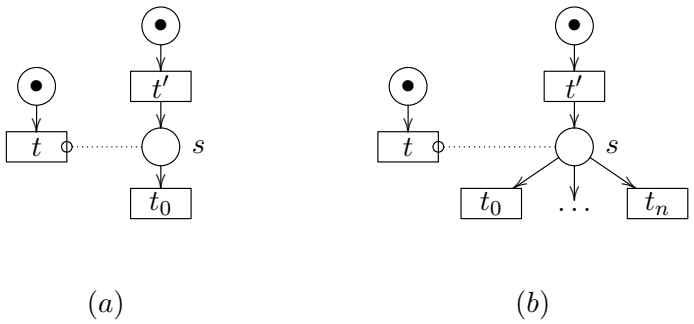


Figure 8. Two basic nets with inhibitor arc.

it is the only relation of a IES.

Remarkably, computations of an inhibitor net (and thus configurations of an IES) involving the same events may differ from the point of view of causality. For instance, in the basic net in Fig. 8.(a) there are two possible orders of execution of transitions t, t' and t_0, namely $t;t';t_0$ and $t';t_0;t$, and while in the first case it is natural to think of t as a cause of t', in the second case we can imagine instead that t_0 (and thus t') causes t. To take into account correctly this further information, both configurations of IES's and processes of inhibitor nets are enriched with a so-called *choice relation* specifying which of the possible partially ordered computations we are referring to.

The unfolding construction for inhibitor nets makes an essential use of the construction already developed for contextual nets. The main problem emerges in the passage from occurrence inhibitor net to IES's: the backward steps is impossible, basically because of complications due to the complex kind of causality expressible in IES's. More technically, the construction associating an inhibitor event structure to an occurrence net is functorial, but does not give rise to a categorical coreflection [3].

Lifting the results to graph grammars

When we finally turn our attention to 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 graph grammar computations and the theory developed for inhibitor nets smoothly lifts, at the price of some technical complications, to grammars. Furthermore, not only the process and the unfolding semantics proposed for a graph grammars are shown to agree, but the theory presented in this paper can be shown to be consistent also with the classical theory of concurrency for DPO grammar in the literature, basically relying on *shift-equivalence*. More specifically:

(i) We define a Winskel's style semantics for graph grammars [6,7]. An unfolding construction is presented, which associates to each graph grammar a nondeterministic occurrence grammar describing its behaviour. Such a construction establishes a coreflection between suitable categories of 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).

(ii) We introduce a notion of *nondeterministic graph process* generalizing the deterministic processes of [16]. The notion fits nicely in our theory since a graph process of a grammar $\mathcal{G}$ is defined simply as a (special kind of) grammar morphism from an occurrence grammar into $\mathcal{G}$ (while in [16] an ad hoc mapping was used).

(iii) We define *concatenable graph processes*, as a variation of (deterministic finite) processes endowed with an operation of concatenation which models sequential composition of computations [5]. The appropriateness of this notion is confirmed by the fact that the category $\mathbf{CP}[\mathcal{G}]$ of concatenable processes of a grammar $\mathcal{G}$ turns out to be isomorphic to the classical truly concurrent model of computation of a grammar based on traces of [17].

(iv) The event structure obtained via the unfolding is shown to coincide both with the one defined in [15] via a comma category construction on the category of concatenable derivation traces, and with the one proposed in [40], based on a deterministic variant of the DPO approach (see [6]). 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 Conclusions

The main result of our work is the development of a systematic theory of concurrency for DPO graph grammars, which contribute to close the gap existing between graph transformation systems and Petri nets. A second achievement is the development of an analogous unifying theory for two widely diffused generalizations of Petri nets, namely *contextual nets* and *inhibitor nets*. While a theory of deterministic processes for these kind of nets was already available in the literature (see, e.g., [32,10]), the Winskel-style semantics, comprising the unfolding construction, its abstraction to a prime algebraic semantics, as well as its relation with the deterministic process semantics are original.

A problem which remains open regards the possibility of fully extending Winskel's construction also to inhibitor nets and graph grammars, by expressing as a coreflection the whole semantical transformation leading from the category of systems to the category of domains. In fact, while the results on contextual nets can be considered completely satisfactory, in the case of inhibitor nets and graph grammars the absence of a coreflection with the category of inhibitor event structures suggests that the construction should still be improved.

An aspect which has not been considered in this paper is the *abstract algebraic characterization* of the model of computation of a system. Well established results exist for ordinary Petri nets, whose computations have been characterized in terms of monoidal categories [29,39]. For graph transformation systems the problem is still unsolved, but some work in the PhD thesis [24] and in [22], suggests the need of resorting to more complex categorical structures like bi- or double categories.

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) [42], which, differently from ordinary bisimulation, takes into account the properties of concurrency of the system. Furthermore, once an unfolding construction has been defined, a natural question suggested by the work initiated

in [28] regards the possibility of extracting from the (possibly infinite) unfolding a finite fragment which is still useful to study some relevant properties of the system. For both problems some encouraging results has been obtained in the case of contextual nets (see, e.g., [8] and [44]).

Finally, although we considered only on basic 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 is trivial, developing a similar theory for more general structures and for abstract categories (e.g., high level replacement systems [21]) 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. 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.

[4] P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. To appear in *Information and Computation*.

[5] 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.

[6] 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.

[7] 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.

[8] P. Baldan, A. Corradini, and U. Montanari. History preserving bisimulations for contextual nets. In D. Bert and C. Choppy, editors, *WADT'99 Conference Proceedings*, number 1827 in LNCS, pages 291–310. Springer Verlag, 2000.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[17] 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.

[18] 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.

[19] P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Informatica*, 33:641–647, 1996.

[20] 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.

[21] 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.

[22] F. Gadducci, R. Heckel, and M. LLabrés. A bicategorical axiomatization of concurrent graph rewriting. In *Proceedings of CTCS'99*, volume 7 of *ENTCS*. Elsevier, 1999.

[23] U. Golz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.

[24] R. Heckel. *Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems.* PhD thesis, TU Berlin, 1998.

[25] R. Janicki and M. Koutny. Invariant semantics of nets with inhibitor arcs. In *Proceedings of CONCUR '91*, volume 527 of *LNCS*. Springer Verlag, 1991.

[26] R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123:1–16, 1995.

[27] H.-J. Kreowski. *Manipulation von Graphmanipulationen.* PhD thesis, Technische Universität Berlin, 1977.

[28] K.L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.

[29] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

[30] 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.

[31] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Place/Transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1997.

[32] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6), 1995.

[33] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.

[34] J.L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[35] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Institutes für Instrumentelle Matematik, Bonn, 1962.

[36] C.A. Petri. Non-sequential processes. Technical Report GMD-ISF-77-5, Gesellshaft für Mathematik und Datenverarbeitung, Bonn, 1977.

[37] W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.

[38] G. Ristori. *Modelling Systems with Shared Resources via Petri Nets.* PhD thesis, Department of Computer Science - University of Pisa, 1994.

[39] V. Sassone. An axiomatization of the algebra of Petri net concatenable processes. *Theoretical Computer Science*, 170:277–296, 1996.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.