Relating SPO and DPO graph rewriting with Petri nets having read, inhibitor and reset arcs

P. Baldan [a] and A. Corradini [b] and U. Montanari [b]

[a] *Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy*

[b] *Dipartimento di Informatica, Università di Pisa, Italy*

Abstract

It belongs to the folklore that graph grammars can be seen as a proper generalisation of Petri nets. In this paper we show how this intuitive relationship can be made formal. The double-pushout approach to graph rewriting turns out to be strictly related to Petri nets with read and inhibitor arcs, while the single-pushout approach has strong connections to Petri nets with read and reset arcs.

Introduction

It belongs to the folklore that graph transformation systems can be seen as a proper generalisation of Petri nets [19,11]. The correspondence is based on the observation that a Petri net is essentially a rewriting system over structures simpler than graphs, i.e., (multi)sets or markings, the rewriting rules being the transitions of the net. And, in turn, net markings can be represented as a restricted kind of graphs, namely discrete, labelled graphs where the presence of a token in a place s is represented by including a node labelled by s.

In this view graph transformation systems are a proper generalisation of Petri nets in two dimensions:

- They allow to specify *contextual* rewritings, namely rewriting steps which can occur only in the presence of a specific context which is *preserved*, i.e., not affected by the step.
- They allow to deal with *general graphs*, rather than discrete graphs.

The possibility of specifying a context for a rewriting step has an interesting computational interpretation: the context can be thought of as a part of the state which is accessed in a read-only manner. This represents the basis to

⋆ Research partially supported by the EU FET – GC Project IST-2001-32747 AGILE and the EC RTN 2-2001-00346 SegraVis.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

establish a connection with *contextual nets* [24], an extension of ordinary Petri nets where transitions can check for the presence of tokens without consuming them.

The second generalisation, i.e., the fact that general graphs are rewritten, leads to complex kinds of dependencies between events in graph grammar computations [4]. In particular, in the DPO approach [16,13], the *dangling condition* prevents a rule to be applied at a given match when its application would leave a dangling edge. The fact that the dangling condition is a negative application condition establishes a link with a further extension of Petri nets, the nets with *inhibitor arcs* [1] where transitions can check for the absence of tokens in places. In the SPO approach [21,17], instead, there is no dangling condition: when a node is deleted by the application of a rule also all the edges connected to such node are deleted by the rewriting step. In this way, the edges which would remain dangling are removed as a kind of *side-effect* of the application of the rule. This is reminiscent of *reset arcs* [3] in the world of Petri nets: a transition connected to a place s through a reset arc can fire independently of the state of s, but its firing will always remove all the tokens in s.

In this paper we show how the intuitive correspondences outlined above can be made formal. Graph transformation systems in the DPO approach turn out to be strictly related to Petri nets with read and inhibitor arcs (*inhibitor nets*), while the SPO approach exhibits strong connections to Petri nets with read and reset arcs (*reset nets*). More precisely we show that any safe DPO grammar can be encoded as a (safe) inhibitor net, while any safe SPO grammar can be encoded as a (safe) reset net. Vice versa, general inhibitor and reset nets can be encoded into DPO and SPO grammars, respectively.

An inhibitor net is encoded as a DPO grammar which acts on possibly non-discrete graphs. Roughly, for any transition t of the original net, if t is inhibited by a place s then the corresponding production p_t deletes (and produces again) a node n_s. Then the encoding ensures that in the graph there are edges attached to n_s if and only if the corresponding place s is marked in the net. In this way, due to the dangling condition, the production p_t will be applicable to all and only the graphs corresponding to the markings where transition t was fireable. The encoding of reset nets into SPO grammars is completely analogous.

All encodings mentioned above preserve the sequential behaviour of the models at hand (firing/rewriting relation and reachability). Furthermore, the encodings defined for safe (DPO and SPO) grammars, as well as those for *safe* inhibitor and reset nets, also preserve the concurrent behaviour of the involved systems. However, the encodings of *general* nets into graph transformation systems are *not* faithful from the point of view of concurrency. Intuitively, this happens because in the encoding of a transition as a DPO (SPO) production, the negative testing (reset) operation is simulated by an action which deletes a part of the state (node n_s) and this imposes a bound to the number of tests which can be conducted in parallel.

1 The Single and Double Pushout Approaches

This section introduces some basics of the algebraic approaches to graph rewriting considered in the paper. We concentrate on *typed hypergraph rewriting systems*, both in the *single-pushout* (SPO) [21,17] and *double-pushout* (DPO) [16,13] approaches. Typed rewriting is a well-established variant of the classical approach where rewriting takes place on so-called typed graphs, i.e., graphs labelled over a structure which is itself a graph [12,22].

1.1 Hypergraphs and hypergraph morphisms

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

Given a set A we denote by A^* the set of finite strings of elements of A (free monoid over A). Given $u \in A^*$ we write $|u|$ to indicate the length of u and u_i to denote the i-th element of u. Furthermore, if $f : A \to B$ is a function then we denote by $f^* : A^* \to B^*$ its extension to strings.

A *(hyper)graph* G is a triple $G = (V_G, E_G, c_G)$, where V_G is a set of nodes, E_G is a set of edges and $c_G : E_G \to V_G^*$ is a function mapping each edge to the list of nodes it is connected to. We will improperly write $n \in c_G(e)$ if node n is connected to edge e, i.e., if $n \in V_G$, $e \in E_G$, and there are $s, s' \in V_G^*$ such that $c_G(e) = sns'$. An edge $e \in E_G$ is called an k-ary edge if $|c_G(e)| = k$. A graph G is called *edge-discrete* if $V_G = \emptyset$ (and thus all edges are 0-ary).

A graph G will be considered sometimes as an unstructured collection of nodes and edges (assuming, without loss of generality, that $V_G \cap E_G = \emptyset$). Thus usual set operations and relations will be applied to graphs.

Definition 1.1 [partial graph morphism] A *partial graph morphism* $f : G \rightarrowtail H$ is a pair of partial functions $f = \langle f_N : N_G \rightarrowtail N_H, f_E : E_G \rightarrowtail E_H \rangle$ such that

(i) for any $e \in E_G$, if $f(e)$ is defined then $|c_G(e)| = |c_H(f(e))|$

(ii) $c_H \circ f_E \leq {f_N}^* \circ c_G$ (see Fig. 1.(a))

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

Notice that if f is defined over an edge then it must be defined on all the nodes the edge is attached to: this ensures that the domain of f is a well-formed graph. The inequality in condition (ii) ensures that *any* subgraph of a graph G can be the domain of a partial morphism $f : G \rightarrowtail H$. Instead,

$$
\begin{tikzcd}
E_G \arrow[r, "f_E"] \arrow[d, "c_G"'] \arrow[dr, phantom, "\geq"] & E_H \arrow[d, "c_H"] \\
N_G \arrow[r, "f_N"'] & N_H
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
{|G_1|} \arrow[rr, "f"] \arrow[dr, "t_{G_1}"'] & \geq & {|G_2|} \arrow[dl, "t_{G_2}"] \\
& T &
\end{tikzcd}
$$

(a) (b)

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

the stronger (apparently natural) condition $c_H \circ f_E = {f_N}^* \circ c_G$ would have imposed f to be defined over an edge whenever it is defined on one of the nodes the edge is attached to.

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

Given a partial typed graph morphism $f : G_1 \rightarrowtail G_2$, we denote by $dom(f)$ the domain of f typed in the obvious way. Since in this paper we work only with typed notions, we will usually omit the qualification "typed".

1.2 DPO rewriting

Fixed a graph T of types, a *(T-typed graph)* DPO *production* $q = (L_q \stackrel{l_q}{\leftarrow} K_q \stackrel{r_q}{\to} R_q)$ is a pair of *injective* typed graph morphisms $l_q : K_q \to L_q$ and $r_q : K_q \to R_q$, where $|L_q|$, $|K_q|$ and $|R_q|$ are finite graphs. The typed graphs L_q, K_q, and R_q are called the *left-hand side*, the *interface*, and the *right-hand side* of the production, respectively.

Definition 1.2 [DPO direct derivation] Given a typed graph G, a production q, and a *match* (i.e., a graph morphism) $g : L_q \to G$, a *direct derivation δ from G to H using q (based on g)* exists, written $\delta : G \Rightarrow_q H$, if and only if the diagram

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

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

Given an injective morphism $l_q : K_q \to L_q$ and a match $g : L_q \to G$ as in the above diagram, their *pushout complement* (i.e., a graph D with morphisms k and b such that the left square is a pushout) exists if and only if the *gluing condition* is satisfied. This consists of two parts:

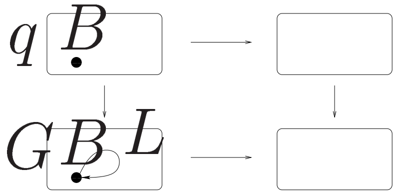

Figure 2. Side-effects in SPO rewriting.

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

1.3 SPO rewriting

Fixed a graph T of types, a *(T-typed graph)* SPO *production* q is an injective partial typed graph morphism $L_q \overset{r_q}{\rightarrowtail} R_q$. The typed graphs L_q and R_q are called the *left-hand side* and the *right-hand side* of the production, respectively.

Definition 1.3 [SPO direct derivation] Given a typed graph G, a production q, and a *match* (i.e., a total graph morphism) $g : L_q \to G$, we say that there is a *direct derivation δ from G to H using q (based on g)*, written $\delta : G \Rightarrow_q H$, if the following is a pushout square in T-**PGraph**.

$$\begin{tikzcd} L_q \arrow[r, tail, "r_q"] \arrow[d, "g"'] & R_q \arrow[d, tail, "h"] \\ G \arrow[r, tail, "d"'] & H \end{tikzcd}$$

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

A relevant difference with respect to the DPO approach is that here there is no *dangling condition* preventing a rule to be applied whenever its application would leave dangling edges. In fact, as a consequence of the way pushouts are constructed in T-**PGraph**, when a node is deleted by the application of a rule also all the edges connected to such node are deleted by the rewriting step, as a kind of *side-effect*. For instance, production q in the top row of Fig. 2, which consumes node B, can be applied to the graph G in the same figure. As a result both node B and the loop edge L are removed.
Even if the category **PGraph** has all pushouts, still we will consider a condition which corresponds to the *identification condition* of the DPO approach.

Definition 1.4 [valid match] A match $g : L_q \to G$ is called *valid* when for any $x, y \in |L_q|$, if $g(x) = g(y)$ then $x, y \in dom(r_q)$.

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

In the paper we will consider only *valid* derivations. This is needed to have a computational interpretation which is resource-conscious, i.e., where a resource can be consumed only once.

1.4 Graph grammars

Graph grammars in the DPO/SPO approach are defined as collections of DPO/SPO productions with a start graph.

Definition 1.5 [typed graph grammar] A *(T-typed) graph grammar* $\mathcal{G}$ is a tuple $\langle T, G_s, P, \pi \rangle$, where G_s is the *start (typed) graph*, P is a set of *production names*, and π is a function which associates a graph production to each production name in P.

We will assume that for each production name q the corresponding production $\pi(q)$ is, in the DPO case, $L_q \stackrel{l_q}{\leftarrow} K_q \stackrel{r_q}{\rightarrow} R_q$, and in the SPO case $L_q \stackrel{r_q}{\rightarrowtail} R_q$.

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

Given an SPO graph grammar, we can easily construct a corresponding DPO grammar, which behaves as the original grammar when the dangling condition is satisfied. Clearly, the converse construction is possible as well.

Definition 1.6 Let $q = L_q \stackrel{r_q}{\rightarrowtail} R_q$ be an SPO production. The corresponding DPO production, denoted by $\mathcal{D}(q)$ is

$$L_q \stackrel{l_q}{\leftarrow} dom(r_q) \stackrel{r_q}{\rightarrow} R_q$$

where l_q is the inclusion of $dom(r_q)$ into L_q and, with abuse of notation, r_q denotes the total function obtained as the restriction of r_q to its domain.

Given a SPO grammar $\mathcal{G} = \langle T, G_s, P, \pi \rangle$, we denote by $\mathcal{D}(\mathcal{G})$ the DPO grammar $\langle T, G_s, P, \pi' \rangle$, where for all $q \in P$, $\pi'(q) = \mathcal{D}(\pi(q))$.

Vice versa, given a DPO grammar $\mathcal{G}$ we can define, in a dual way, the SPO grammar $\mathcal{S}(\mathcal{G})$.

For instance, if we consider the DPO grammar $\mathcal{G}_2$ in Fig. 3 and the SPO grammar $\mathcal{G}_2'$ in Fig. 4 then $\mathcal{G}_2 = \mathcal{S}(\mathcal{G}_2')$ and $\mathcal{G}_2' = \mathcal{D}(\mathcal{G}_2)$.

2 Petri nets with read, inhibitor and reset arcs

In this section we introduce some basic extensions of P/T Petri nets, i.e., Petri nets with read, inhibitor and reset arcs. As already mentioned *read arcs* (also called test, activator or positive contextual arcs) [9,24,18,27] allow a transition to check for the presence of tokens without consuming them. This allows to represent faithfully the situations where a resource is read but not consumed (read-only accesses). Read arcs have been used, for example, to model concurrent accesses to shared data (e.g., read operations in a database) [26,14], to study temporal efficiency in asynchronous systems [27] and to give a truly concurrent semantics to concurrent constraint programs [23,6].

Inhibitor arcs, which allow a transition to test for the *absence of tokens* in a place, have been introduced in [1] to solve a synchronisation problem not expressible in classical Petri nets. This extension makes the model Turing complete because they allow to simulate the zero-testing operation of RAM machines which cannot be expressed neither by flow nor by read arcs. Inhibitor arcs have been employed, for example, for performance evaluation of distributed systems [2], to provide π-calculus with a net-based semantics [7] and to show the existence of an expressiveness gap between two different semantics of a process algebra based on Linda coordination primitives [8].

Reset arcs [3] allow a transition to remove all the tokens in a place, if any: the notion of enabling is not influenced by reset arcs, but if a transition is connected to a place by a reset arc then its firing *resets* the content of the place. Reset arcs have been used, e.g., to model communications through unreliable channels where messages can be lost (see [5,10] and references therein). It is worth stressing that nets with reset arcs have peculiar decidability properties: reachability and boundedness are undecidable, while coverability and termination are decidable [15]. As such they lie somehow between ordinary nets and nets with inhibitor arcs.

A study of the expressiveness of these kinds of arcs, along with a comparison with other extensions proposed in the literature, like priorities, exclusive-or transitions and switches, is carried out in [25,20].

To give the formal definition of these generalised Petri nets we need some notation for sets and multisets. Let A be a set. The powerset of A is denoted by $\mathbf{2}^A$. A *multiset* of A is a function $M : A \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. The set of multisets of A is denoted by μA. The usual operations and relations on multisets, like multiset union $+$ or multiset difference $-$, are used. We write $M \leq M'$ if $M(a) \leq M'(a)$ for all $a \in A$. If $M \in \mu A$, we denote by $[\![M]\!]$ the multiset defined as $[\![M]\!](a) = 1$ if $M(a) > 0$ and $[\![M]\!](a) = 0$ otherwise, obtained by changing all non-zero coefficients of M to 1; sometimes $[\![M]\!]$ will be confused with the corresponding subset $\{a \in A \mid [\![M]\!](a) = 1\}$ of A. Given a multiset M and a subset $X \subseteq A$, the multiset $M' = M \downarrow X$ is defined by $M'(a) = M(a)$ if $a \notin X$ and $M'(a) = 0$, otherwise.

A *multirelation* $f : A \to B$ is a multiset of $A \times B$. We will limit our attention to finitary multirelations, namely multirelations f such that the set $\{b \in B \mid f(a,b) > 0\}$ is finite. Multirelation f induces in an obvious way a (possibly partial) function $\mu f : \mu A \to \mu B$, defined as $\mu f(\sum_{a \in A} n_a \cdot a) =$

$\sum_{b\in B}\sum_{a\in A}(n_a \cdot f(a,b)) \cdot b$.[1]

Definition 2.1 [read, inhibitor and reset arcs] A *(marked) Petri net* with read, inhibitor and reset arcs is a tuple $N = \langle S, Tr, F, C, I, R, m\rangle$, where

- S is a set of *places*;
- Tr is a set of *transitions*;
- $F = \langle F_{pre}, F_{post}\rangle$ is a pair of multirelations from Tr to S;
- C, I and R are relations between Tr and S, called the *context*, *inhibitor* and *reset relation*, respectively;
- $m \in \mu S$ is a multiset called the *initial marking*.

Some subclasses of nets will play a relevant role in the paper:

- If C, I and R are empty, the net is called *ordinary net* and denoted by $N_O = \langle S, Tr, F, m\rangle$.
- If I and R are empty, the net is called *contextual net* and denoted by $N_C = \langle S, Tr, F, C, m\rangle$.
- If R is empty, the net is called *inhibitor net* and denoted by $N_I = \langle S, Tr, F, C, I, m\rangle$.
- If I is empty, the net is called *reset net* and denoted by $N_R = \langle S, Tr, F, C, R, m\rangle$.

We assume, as usual, that $S \cap Tr = \emptyset$. The functions from μTr to μS induced by the multirelations F_{pre} and F_{post} are denoted by ${}^\bullet(\ )$ and $(\ )^\bullet$, respectively. If $A \in \mu Tr$ is a multiset of transitions, ${}^\bullet A$ is called its *pre-set*, while $A^\bullet$ is called its *post-set*. Moreover, by $\underline{A}$ we denote the *context* of A, defined as $\underline{A} = C([\![A]\!])$, by ${}^{\circledcirc}A = I([\![A]\!])$ the *inhibitor set* of A, and by ${}^{\ominus}A = R([\![A]\!])$ the *reset set* of A. The same notation is used to denote the functions from S to $\mathbf{2}^{Tr}$ defined as, for $s \in S$, ${}^\bullet s = \{t \in Tr \mid F_{post}(t,s) > 0\}$, $s^\bullet = \{t \in Tr \mid F_{pre}(t,s) > 0\}$, $\underline{s} = \{t \in Tr \mid C(t,s)\}$, ${}^{\circledcirc}s = \{t \in Tr \mid I(t,s)\}$ and ${}^{\ominus}s = \{t \in Tr \mid R(t,s)\}$.

A finite multiset of transitions A is enabled at a marking M, if M contains the pre-set of A and an additional set of tokens which covers the context of A. Furthermore the places of the inhibitor set of A must be empty both before and after the firing of the transitions in A. Finally, no place in the reset set of A can be used (read, consumed or produced).

Definition 2.2 [token game] Let N be a net with read, inhibitor and reset arcs, and let M be a *marking* of N, i.e., a multiset $M \in \mu S$. A finite multiset $A \in \mu Tr$ is *enabled* at M if (i) ${}^\bullet A + \underline{A} \leq M$, (ii) $[\![M + A^\bullet]\!] \cap {}^{\circledcirc}A = \emptyset$, and (iii) $({}^\bullet A + \underline{A} + A^\bullet) \cap {}^{\ominus}A = \emptyset$. The *transition relation* between markings is defined as

[1] The function μf can be partial since infinite coefficients are disallowed in multisets. For instance, given the multirelation $f : \mathbb{N} \to \{0\}$ with $f(n,0) = 1$ for all $n \in \mathbb{N}$, then μf is undefined on the multiset $\sum_{n\in\mathbb{N}} 1 \cdot n$.

$$M\,[A\rangle\,M' \qquad \text{iff} \qquad A \text{ is enabled at } M \text{ and } M' = (M - {}^{\bullet}A + A^{\bullet}) \downarrow {}^{\ominus}A.$$

Step and firing sequences, as well as reachable markings, are defined in the usual way.

3 Encoding safe grammars as extended nets

We present here an encoding of graph grammars as extended nets. The encoding works for a restricted class of grammars only, namely for *safe* grammars. The notion of safeness for grammars, originally defined in the context of DPO [12], generalises the corresponding notion for P/T nets, which requires that each place contains at most one token in any reachable marking. Since large part of the theory is independent from the adopted approach, when possible we will give notions and constructions for a generic grammar $\mathcal{G}$, without sticking specifically to the DPO or SPO approach.

Definition 3.1 [safe grammar] A grammar $\mathcal{G} = \langle T, G_s, P, \pi \rangle$ is *safe* if, for all H such that $G_s \Rightarrow^* H$, H has an injective typing morphism.

The definition can be understood by thinking of nodes and edges of the type graph as a generalisation of places in Petri nets. In this view the number of different items of a graph which are typed on a given item of the type graph corresponds to the number of tokens contained in a place, and thus the condition of safeness for a marking is generalised to typed graphs by the injectivity of the typing morphism.

Safe graph grammars admit a natural net-like pictorial representation, where items of the type graph and productions play, respectively, the role of places and transitions of Petri nets. The basic observation is that typed graphs having an injective typing morphism can be safely identified with the corresponding subgraphs of the type graph (just thinking of injective morphisms as inclusions). Therefore, in particular, each graph $G = \langle |G|, t_G \rangle$ reachable in a safe grammar can be identified with the subgraph $t_G(|G|)$ of the type graph T, and thus it can be represented by suitably decorating the nodes and edges of T. Concretely, nodes and edges are drawn as circles and boxes, respectively, and they are filled if they belongs to $t_G(|G|)$ and empty otherwise (see Figure 3). This is analogous to the usual technique of representing the marking of a safe net by putting a token in each place which belongs to the marking.

With the above identification, in each derivation of a safe grammar starting from the start graph a production can be applied only to the subgraph of the type graph which is the image via the typing morphism of its left-hand side. Therefore according to its typing, we can think that a production *produces*, *preserves* and *consumes* items of the type graph. This is expressed by drawing productions as arrow-shaped boxes, connected to the consumed and produced resources by incoming and outcoming arrows, respectively, and to the preserved resources by undirected lines.

For instance, Fig. 3 presents two examples of DPO safe grammars, with their pictorial representation. Notice that the typing morphisms for the start graph and the productions are represented by suitably labelling the involved graphs with items of the type graph.

A completely analogous representation can be given for SPO grammars. Formally, we represent any SPO grammar $\mathcal{G}$ exactly as the corresponding DPO grammar $\mathcal{D}(\mathcal{G})$. For instance, the SPO grammar $\mathcal{G}_2'$ in Fig. 4 is represented exactly as the grammar $\mathcal{G}_2$ in Fig. 3, since $\mathcal{D}(\mathcal{G}_2') = \mathcal{G}_2$.

Given the graphical representation of safe DPO and SPO grammars, as in the right part of Fig. 3, one could play the token game on them as if they were contextual Petri nets. However, the resuting behaviour would not correspond faithfully to the behaviour of the original grammars, since the token game can delete and generate nodes and edges freely, without any care for the consistency of the graphical structure of the state. In the next two subsections we will show how the encoding can be finalised, separately for DPO and SPO grammars, by adding inhibitor and reset arcs, respectively.

Before that, we will define the *pre-set* ${}^\bullet q$, *context* $\underline{q}$ and *post-set* $q^\bullet$ of a production q, thus introducing a net-like language for the productions of a grammar. These notions have a clear interpretation for safe grammars only. However, for technical reasons we define them for arbitrary graph grammars.

Definition 3.2 [pre-set, post-set, context]

(i) Let $\mathcal{G}$ be a DPO graph grammar. For any $q \in P$ we define

$${}^\bullet q = t_{L_q}(|L_q| - l_q(|K_q|)) \qquad q^\bullet = t_{R_q}(|R_q| - r_q(|K_q|)) \qquad \underline{q} = t_{K_q}(|K_q|)$$

seen as sets of nodes and edges of the graph of types, and we say that q *consumes*, *creates* and *preserves* items in ${}^\bullet q$, $q^\bullet$ and $\underline{q}$, respectively.

(ii) The definition in the case of SPO grammars is essentially the same: the pre-set, post-set and context of a production $q = r_q : L_q \rightarrowtail R_q$ are defined as above just replacing $dom(r_q)$ for $l_q(|K_q|)$ and $r_q(dom(r_q))$ for $r_q(|K_q|)$.

For instance, for the DPO grammar $\mathcal{G}_2$ in Figure 3, as well as for the SPO grammar $\mathcal{G}_2'$ in Figure 4, the pre-set, context and post-set of production p_1 are ${}^\bullet p_1 = \{C\}$, $\underline{p_1} = \{B\}$ and ${p_1}^\bullet = \{A, L\}$.

3.1 Encoding in the DPO case

In the case of DPO grammars, because of the dangling condition, a production q which consumes a node n can be applied only if there are no edges connected to n which would remain dangling after the application of q. In other words, if $n \in {}^\bullet q$, $e \notin {}^\bullet q$ and $n \in c(e)$ then the application of q is *inhibited* by the presence of e. By analogy with inhibitor nets we introduce the *inhibitor set* of a production.

Definition 3.3 [inhibitor set] Let $\mathcal{G}$ be a DPO graph grammar. The *inhibitor set* of a production $q \in P$ is defined by

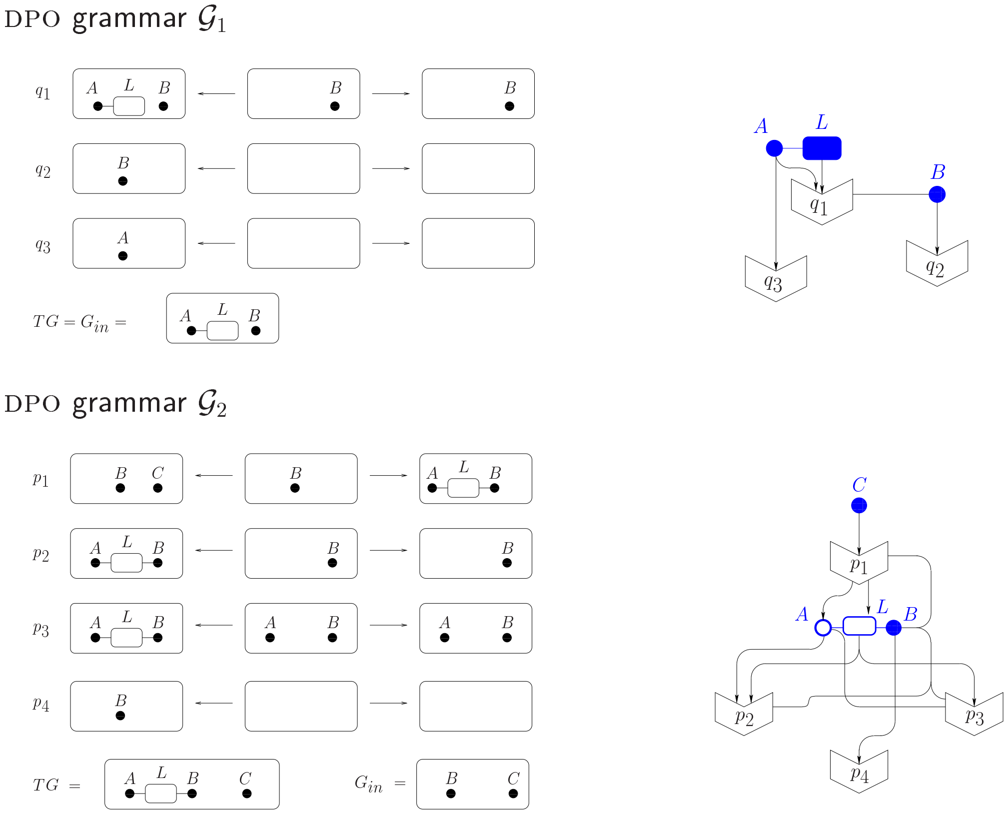

Figure 3. Two safe DPO grammars and their net-like representation.

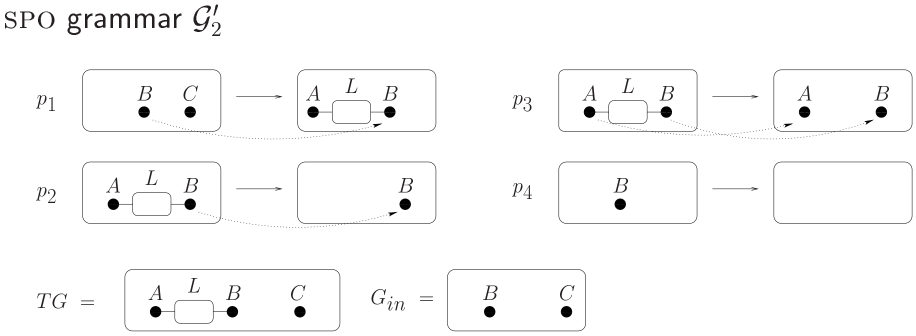

Figure 4. An SPO grammar corresponding to the DPO grammar $\mathcal{G}_2$ in Fig.3. The partial morphisms from the lhs to the rhs of productions are represented by the dotted arrows.

$$^{\circledcirc}q = \{e \in E_T \mid \exists n \in {}^{\bullet}q.\ n \in c(e) \ \wedge \ e \not\in {}^{\bullet}q\}$$

For instance, in the grammar $\mathcal{G}_2$ of Figure 3 the inhibitor set of p_4 is $^{\circledcirc}p_4 = \{L\}$.

Note that with the above definition, a production q of a safe grammar $\mathcal{G}$ satisfies the dangling condition w.r.t. the match determined by the typing morphism if and only if $t_{L_q}(|L_q|) \cap {}^{\circledcirc}q = \emptyset$.

The correspondence between DPO safe grammars and inhibitor nets can be

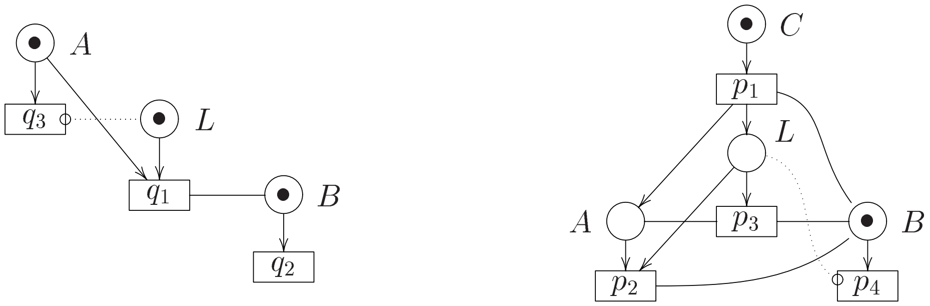

Figure 5. The inhibitor nets $\mathcal{N}_I(\mathcal{G}_1)$ and $\mathcal{N}_I(\mathcal{G}_2)$ corresponding to the grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 3.

made more explicit:

Definition 3.4 [encoding DPO grammars into inhibitor nets] To any safe DPO grammar $\mathcal{G} = \langle T, G_s, P, \pi \rangle$ we associate an inhibitor net $\mathcal{N}_I(\mathcal{G}) = \langle S, Tr, F, C, I, m \rangle$, where the places are the items of T (i.e., $S = V_T \cup E_T$), the transitions are the productions (i.e., $Tr = P$), and for each transition $t \in Tr$, the pre-set, post-set, context and inhibitor set are defined as in Definitons 3.2(i) and 3.3. The initial marking m is the set of items in (the image in T of) the start graph G_s.

Figure 5 shows the inhibitor nets $\mathcal{N}_I(\mathcal{G}_1)$ and $\mathcal{N}_I(\mathcal{G}_2)$ corresponding to the safe DPO grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ of Figure 3.

It is possible to show that a T-typed DPO grammar $\mathcal{G}$ and the net $\mathcal{N}_I(\mathcal{G})$, as defined above, have essentially the same behaviour in the sense that each (concurrent) derivation in $\mathcal{G}$ corresponds to a step sequence in $\mathcal{N}_I(\mathcal{G})$ using the same productions and leading to the same state, and vice versa. More precisely, given a graph G injectively typed over T, let us denote by $\mathbf{m}(G)$ the corresponding safe marking over the net $\mathcal{N}_I(\mathcal{G})$, i.e., the set of nodes and edges in $t_G(|G|)$. Then the following result holds:

Proposition 3.5 *Let $\mathcal{G}$ be a* DPO *T-typed graph grammar and let G be an injectively T-typed graph. Then there is a parallel direct derivation $G \stackrel{q_1+\ldots+q_n}{\Longrightarrow} G'$ in $\mathcal{G}$ iff there is a step $\mathbf{m}(G)\, [q_1 \oplus \ldots \oplus q_n\rangle\, \mathbf{m}(G')$ in $\mathcal{N}_I(\mathcal{G})$.*

3.2 Encoding in the SPO case

The encoding for SPO grammars is formally the same as for DPO grammars, but with inhibitor arcs replaced by reset arcs. In fact, as already mentioned, when a production q which deletes a node n is applied, all the edges connected to n that would remain dangling after the application of q are removed. Therefore, if $n \in {}^\bullet q$, $e \notin {}^\bullet q$ and $n \in c(e)$, then we can say that the application of q *resets* the edge e. By analogy with reset netsm we introduce the *reset set* of a production.

Definition 3.6 [reset set] Let $\mathcal{G}$ be a SPO graph grammar. The *reset set* of a

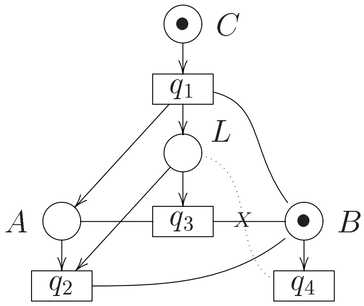

Figure 6. The reset nets corresponding to the grammar $\mathcal{G}'_2$ in Fig. 4.

production $q \in P$ is defined by

$$^{\ominus}q = \{e \in E_T \mid \exists n \in {}^{\bullet}q.\ n \in c(e) \ \wedge\ e \notin {}^{\bullet}q\}$$

For instance, in the SPO grammar $\mathcal{G}'_2$ of Figure 4 the reset set of p_4 is $^{\ominus}p_4 = \{L\}$.

The correspondence between safe grammars and inhibitor nets is formalised in the following definition.

Definition 3.7 [encoding SPO grammars into reset nets] To any safe SPO grammar $\mathcal{G} = \langle T, G_s, P, \pi\rangle$ we associate a reset net $\mathcal{N}_R(\mathcal{G}) = \langle S, Tr, F, C, R, m\rangle$, where $S = V_T \cup E_T$, $Tr = P$, and for each transition $t \in Tr$, the pre-set, post-set, context and reset set are defined as in Definitions 3.2(ii) and 3.6. The initial marking m is the set of items in the start graph G_s.

Figure 6 shows the reset net corresponding to the safe SPO grammars $\mathcal{G}'_2$ of Fig. 4. Since $\mathcal{G}_2 = \mathcal{D}(\mathcal{G}'_2)$, this net is obtained simply by replacing inhibitor arcs with reset arcs in the net $\mathcal{N}_I(\mathcal{G}_2)$, depicted in the right side of Fig. 5.

It is possible to show that, as for DPO grammars, a T-typed SPO grammar $\mathcal{G}$ and the reset net $\mathcal{N}_R(\mathcal{G})$, as defined above, have essentially the same behaviour.

Proposition 3.8 *Let $\mathcal{G}$ be a SPO T-typed graph grammar and let G be an injectively T-typed graph. Then there is a parallel direct derivation $G \stackrel{q_1+\ldots+q_n}{\Longrightarrow} G'$ in $\mathcal{G}$ iff there is a step $\mathbf{m}(G)\,[q_1 \oplus \ldots \oplus q_n\rangle\, \mathbf{m}(G')$ in $\mathcal{N}_R(\mathcal{G})$.*

4 Encoding extended nets as graph grammars

We have seen that safe DPO and SPO grammars can be encoded as safe reset and inhibitor nets, respectively. In this section we investigate the possibility of providing an encoding in the opposite direction, i.e., of encoding inhibitor nets into DPO grammars and reset nets into SPO grammars.

4.1 Ordinary nets

It is part of the folklore (see e.g.,the discussion in [11] and references therein), that Petri nets can be seen as a very special kind of graph grammars. The simple idea is that the marking of a net can be represented as a discrete graph typed over the places: a token in place s is represented as a node typed over s. Then transitions are seen as productions which consume and produce nodes, as prescribed by their pre- and post-set. In this way, Petri nets exactly correspond to graph grammars acting over discrete graphs, where productions do not preserve items.

To make it extensible to more general classes of Petri nets, here we consider a slightly different encoding, where edges, rather than nodes, play the role of tokens, i.e., Petri nets are seen as grammars acting on *edge-discrete graphs*.

The precise way in which the encoding of markings into graphs is defined will depend on the specific kind of nets we are considering. However, in all cases there will be an edge s in the type graph for each place s in the net, and the number of edges typed over s will represent the number of tokens in that place.

Definition 4.1 [relating markings and typed graphs] Let G by a T-typed graph. The marking generated by G, denoted by $\mathbf{m}(G) \in \mu E_T$, is defined by $\mathbf{m}(G)(e) = |t_G^{-1}(e)|$ for any $e \in E_T$.

Given a graph T, any marking $m \in \mu E_T$ corresponds to a T-typed graph $\mathbf{G}_T(m)$ defined as follows:

- $|\mathbf{G}_T(m)| = \langle V, E, c \rangle$, where
  - $V = \{n \in V_T \mid \exists e \in E_T . n \in c_T(e) \wedge m(e) > 0\}$, i.e., $|\mathbf{G}_T(m)|$ has nodes taken from T and a node is included if it is attached to an edge which appears in the multiset.
  - $E = \{(e, i) \mid e \in E_T,\ 1 \leq i \leq m(e)\}$
  - for any $(e, i) \in E$, $c(e, i) = c_T(e)$.
- The typing $t_{\mathbf{G}_T(m)}$ is the identity on nodes, and $t_{\mathbf{G}_T(m)}(e, i) = e$ for all $(e, i) \in E$.

Observe that, since isolated nodes do not contribute to the marking, given a graph T and a marking $m \in \mu E_T$, the graph $\mathbf{G}_T(m)$ is, in some sense, the least T-typed graph such that $\mathbf{m}(\mathbf{G}_T(m)) = m$.

Definition 4.2 Let $N_O = \langle S, Tr, F, m \rangle$ be an ordinary Petri net. The corresponding DPO grammar $\mathcal{G}_d(N_O) = \langle T, G_s, P, \pi \rangle$ is defined as follows.

- $T = (\emptyset, S, c_\epsilon)$, where $c_\epsilon(s) = \epsilon$ (the empty string) for all $s \in S$
- $P = Tr$
- $G_s = \mathbf{G}_T(m)$
- for any $t \in P$, $\pi(t) = \mathbf{G}_T({}^\bullet t) \leftarrow \emptyset \rightarrow \mathbf{G}_T(t^\bullet)$.

Observe that, as anticipated, the type graph T is an "edge-discrete" hypergraph, with only 0-ary edges, one for each place. Production names are net transitions, and the span associated with each production is the obvious one, with empty interface since nothing is explicitly preserved by a P/T net transition.

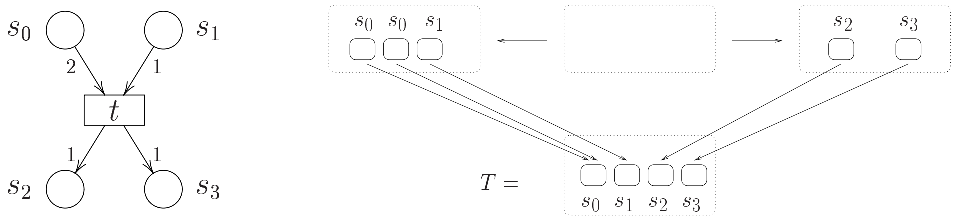

Figure 7. A Petri net transition and the corresponding DPO production.

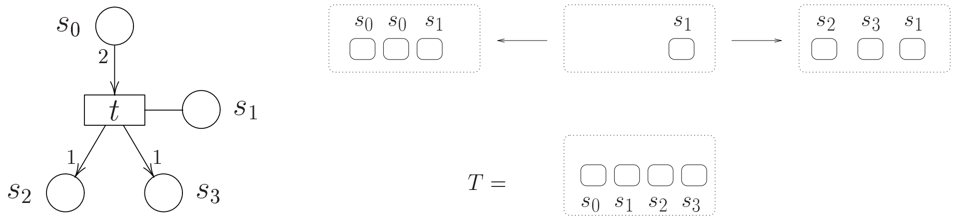

Figure 8. A contextual Petri net transition and the corresponding DPO production.

The encoding as SPO grammar is completely analogous. We can define the SPO grammar corresponding to a net N_O as $\mathcal{G}_s(N_O) = \mathcal{S}(\mathcal{G}_d(N_O))$.

In Fig. 7 the reader can find a Petri net transition and the corresponding DPO production, with the typing morphism depicted explicitly.

4.2 Contextual nets

The encoding of contextual nets as graph grammars is an easy extension of the previous one. The corresponding grammars still operate only on edge-discrete graphs. The novelty is that, since a transition of a contextual net can "read" some token via read arcs, the corresponding productions can have a non-empty interface, i.e., they can preserve some edges.

Definition 4.3 Let $N_C = \langle S, Tr, F, C, m\rangle$ be a contextual Petri net. The corresponding DPO grammar $\mathcal{G}_d(N_C) = \langle T, G_s, P, \pi\rangle$ is defined as follows.

- $T = (\emptyset, S, c_\epsilon)$
- $P = Tr$
- $G_s = \mathbf{G}_T(m)$
- for any $t \in P$, $\pi(t) = \mathbf{G}_T(\underline{t} \oplus {}^\bullet t) \leftarrow \mathbf{G}_T(\underline{t}) \rightarrow \mathbf{G}_T(\underline{t} \oplus t^\bullet)$, where the left and right morphisms in the span are inclusions.

An example of contextual net transition with the corresponding DPO production can be found in Fig. 8. The typing over T is implicitly represented by labelling the items.

As in the previous case, a completely analogous encoding can be carried out using SPO grammars.

4.3 Inhibitor nets

Not surprisingly, inhibitor nets are represented as DPO grammars by using the effects of the dangling condition to encode inhibitor arcs. This requires to move from edge-discrete graphs to graphs which possibly include nodes. More precisely, as in the previous cases the type graph includes one edge for each place s of the net. In addition, if place s inhibits at least one transition, then the type graph contains one node to which the edge corresponding to s is connected. If place s inhibits transition t, the production corresponding to t will delete and produce again the node corresponding to s. In this way the presence of a token in place s, represented by an edge labelled over s connected to such node, will inhibit the production because of the dangling condition.

All the nodes corresponding to inhibiting places are included in the start graph and, since each time they are deleted they are immediately regenerated, they will be present in any reachable graph. As a consequence, the reachable graph corresponding to a marking m will coincide with $\mathbf{G}_T(m)$ only up to isolated nodes. This is formalized by the following definition of complete graph for a marking.

Definition 4.4 [complete graph for a marking] Let T be a graph and let $m \in \mu E_T$ be a marking. The *complete T-typed graph* corresponding to m, denoted $\widetilde{\mathbf{G}}_T(m)$, is defined as $\mathbf{G}_T(m)$ in Definition 4.1, but for the set of nodes of $|\widetilde{\mathbf{G}}_T(m)|$ which is defined as $V_{|\tilde{\mathbf{G}}_T(m)|} = V_T$.

Definition 4.5 Let $N_I = \langle S, Tr, F, C, I, m\rangle$ be an inhibitor Petri net. The corresponding DPO grammar $\mathcal{G}_d(N_I) = \langle T, G_s, P, \pi\rangle$ is defined as follows.

- $T = (S_I, S, c)$, where $S_I = \{s \in S \mid \exists t \in Tr.\ I(t, s)\}$, $c(s) = s$ for each $s \in S_I$, and $c(s) = \epsilon$ for each $s \in S/S_I$
- $P = Tr$
- $G_s = \widetilde{\mathbf{G}}_T(m)$
- for each $t \in P$, $\pi(t) = \mathbf{G}_T(\underline{t} \oplus {}^\bullet t) \uplus G_I \leftarrow \mathbf{G}_T(\underline{t}) \rightarrow \mathbf{G}_T(\underline{t} \oplus t^\bullet) \uplus G_I$, where $G_I = \langle {}^\circledcirc t, \emptyset, \emptyset\rangle$ and the left and right morphisms in the span are inclusions.

Observe that, as mentioned above, the production corresponding to a transition t deletes the edges in its pre-set (preserving the attached nodes), preserves the edges in its context (preserving the attached nodes), produces the edges in its post-set (preserving the attached nodes) and deletes and produces again each node s such that $s \in {}^\circledcirc t$. The last point is ensured by including the graph G_I, which contains exactly such nodes, both in the lhs and rhs, but not in the interface of the production.

A very simple inhibitor net and the corresponding SPO grammar are presented in Fig. 9.

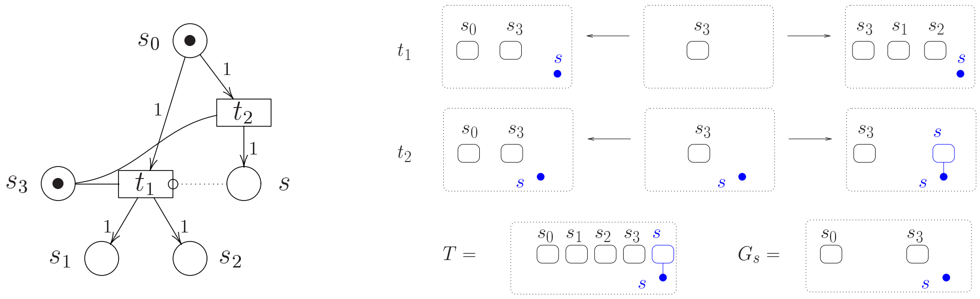

Figure 9. An inhibitor Petri net and the corresponding DPO grammar.

4.4 Reset nets

The duality between DPO/inhibitor arcs and SPO/reset arcs is confirmed also in the encoding of reset nets into SPO grammars. In fact, it can be obtained from the encoding described in the previous section in a trivial way: just replace inhibitor arcs with reset arcs and DPO productions with the corresponding SPO productions.

As before, the type graph contains one edge for each place, and, in addition, for each place which is reset by at least one transition the type graph includes one node, to which the corresponding edge is connected. If transition t resets place s, the production corresponding to t will delete and produce again the node corresponding to s. In this way, all tokens in place s, represented by edges labelled over s connected to that node, will be removed by the application of the corresponding SPO production.

Definition 4.6 Let $N_R = \langle S, Tr, F, C, R, m\rangle$ be a reset Petri net. The corresponding SPO grammar $\mathcal{G}_s(N_R) = \langle T, G_s, P, \pi\rangle$ is defined as follows.

- $T = (S_R, S, c)$, where $S_R = \{s \in S \mid \exists t \in T.\ R(t,s)\}$, $c(s) = s$ for each $s \in S_R$, and $c(s) = \epsilon$ for each $s \in S/S_R$
- $P = Tr$
- $G_s = \widetilde{\mathbf{G}}_T(m)$
- for each $t \in P$, $\pi(t) = \mathbf{G}_T(\underline{t} \oplus {}^{\bullet}t) \uplus G_R \to \mathbf{G}_T(\underline{t} \oplus t^{\bullet}) \uplus G_R$, where $G_R = \langle {}^{\ominus}t, \emptyset, \emptyset\rangle$ and the morphisms is the partial inclusion.

Alternatively, one could define the SPO grammar $\mathcal{G}_s(N_R)$ as $\mathcal{S}(\mathcal{G}_d(N_I))$, where $N_I = \langle S, T, F, C, I, m\rangle$, with $I = R$ is the inhibitor net obtained from N_R replacing reset arcs with read arcs. An example, obtained in this way from the one in Fig. 9, can be found in Fig. 10.

4.5 Correspondence results

The encodings of inhibitor and reset nets into DPO and SPO grammars presented in the previous sections preserve the firing relation and reachability, in the following sense.

Proposition 4.7 *Let N_I be an inhibitor net and let M be a marking of N_I.*

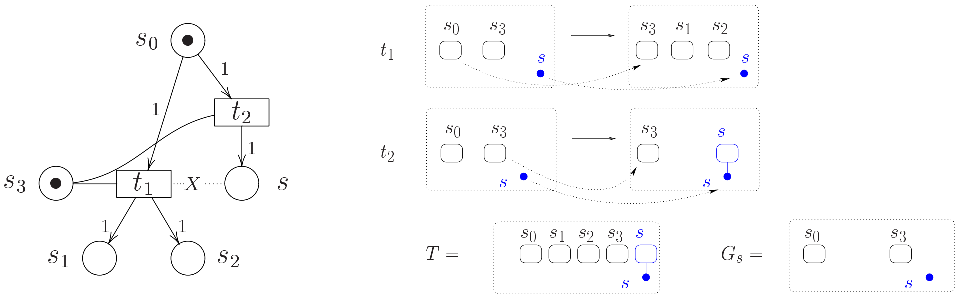

Figure 10. A reset Petri net and the corresponding SPO grammar.

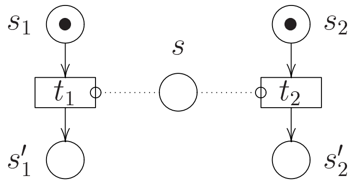

Figure 11. An inhibitor Petri net N_I where transitions can fire concurrently, while in $\mathcal{G}_d(N_I)$ they cannot.

Then $M\,[t\rangle\, M'$ *in* N_I *iff* $\widetilde{\mathbf{G}}_T(M) \stackrel{t}{\Longrightarrow} G'$ *and* $\mathbf{m}(G') = M'$ *in* $\mathcal{G}_d(N_I)$.

Similarly, if N_R *is a reset net and* M *is a marking of* N_R*, then* $M\,[t\rangle\, M'$ *in* N_R *iff* $\widetilde{\mathbf{G}}_T(M) \stackrel{t}{\Longrightarrow} G'$ *and* $\mathbf{m}(G') = M'$ *in* $\mathcal{G}_s(N_R)$.

Unfortunately, the encodings do not preserve the concurrency of the original net. For example, for inhibitor nets, if two transitions are inhibited by the same place s, then their encodings as DPO productions cannot be executed in parallel, since both such productions delete and produce again the node corresponding to s. For instance, in the inhibitor net in Fig. 11, the two transitions t_1 and t_2 can fire concurrently. However, in the corresponding DPO grammar the productions associated to t_1 and t_2 delete and generate again the same node s and thus they are forced to be executed sequentially.

5 Encoding safe nets

If we restrict to safe inhibitor and reset nets, the encodings given in the previous section can be improved to preserve also the concurrent operational behaviour of the original net. We discuss only the case of inhibitor nets, since the one of reset nets is, again, related to it by the usual form of duality.

Given a safe inhibitor net, the idea is to include in the type graph a different node for each transition which is inhibited by a given place. More precisely, for each transition t and for each place s which inhibits t, the type graph includes a node (s, t). There is still only one edge associated to s, and it is connected to all such nodes. Then the production corresponding to t will delete and produce again node (s, t). In this way, because of the dangling condition it will not be enabled if there is a token in s (represented by an s-edge connected to all the (s, t_i) nodes). But having a different node for each transition inhibited by s

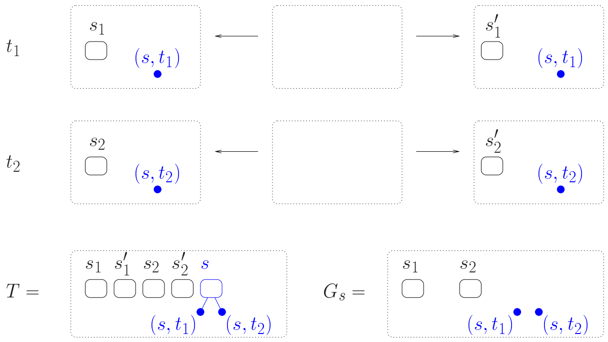

Figure 12. The DPO grammar corresponding to the safe inhibitor net in Fig. 11.

allows one to fire in parallel several transitions inhibited by the same place. Having only one node for each transition of this kind suffices, since in a safe net one cannot have two copies of the same transition concurrently enabled.

Definition 5.1 Let $N_I = \langle S, Tr, F, C, I, m\rangle$ be an inhibitor Petri net. The corresponding DPO grammar $\mathcal{G}_d(N_I) = \langle T, G_s, P, \pi\rangle$ [2] is defined as follows.

- $T = (V, E, c)$, where
  - $V = \bigcup_{t \in Tr} {}^{\circledcirc}t \times \{t\}$,
  - $E = S$
  - $c(s) = (s, t_1) \ldots (s, t_k)$ if ${}^{\circledcirc}s = \{t_1, \ldots .t_n\}$
- $P = Tr$
- $G_S = \widetilde{\mathbf{G}}_T(m)$
- for any $t \in P$, $\pi(t) = \mathbf{G}_T(\underline{t} \oplus {}^{\bullet}t) \uplus G_I \leftarrow \mathbf{G}_T(\underline{t}) \rightarrow \mathbf{G}_T(\underline{t} \oplus t^{\bullet}) \uplus G_I$, where $G_I = \langle {}^{\circledcirc}t \times \{t\}, \emptyset, \emptyset\rangle$ and the left and right morphisms in the span are inclusions.

Informally, the production corresponding to t deletes the edges in its pre-set (preserving the attached nodes), preserve the edges in its context (preserving the attached nodes), produces the edges in its post-set (preserving the attached nodes), deletes and produces again the nodes (t, s) for any $s \in {}^{\circledcirc}t$.

As an example, the DPO graph grammar corresponding to the safe inhibitor net in Fig. 11, can be found in Fig. 12. It can be shown that this new encoding preserve also concurrent steps of the original net.

Proposition 5.2 *Let N_I be a* safe *inhibitor net and let M be a safe marking of N_I. Then $M\,[t_1 + \ldots + t_n\rangle\, M'$ in N_I iff $\widetilde{\mathbf{G}}_T(M) \stackrel{t_1 + \ldots + t_n}{\Longrightarrow} G'$ and $\mathbf{m}(G') = M'$ in $\mathcal{G}_d(N_I)$.*

Proof Just observe that since the marking m is safe, necessarily $t_i \neq t_j$ for $i \neq j$. □

[2] Forse è meglio cambiare simbolo per questo encoding rispetto ai precedenti.

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] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley, 1995.

[3] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3:85–104, 1977.

[4] 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.

[5] J. Billington. *Extensions to Coloured Petri nets and their applications to Protocols.* PhD thesis, University of Cambridge, 1991.

[6] 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.

[7] N. Busi and R. Gorrieri. A Petri Nets Semantics for π-calculus. In *Proceedings of CONCUR'95*, volume 962 of *LNCS*, pages 145–159. Springer Verlag, 1995.

[8] N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of Linda coordination primitives. *Information and Computation*, 2000. To appear.

[9] 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.

[10] G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In R. Valette, editor, *Proceddings of ICAPTN'94*, volume 815 of *LNCS*, pages 179–198. Springer Verlag, 1994.

[11] 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.

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

[13] 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.

[14] 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.

[15] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *Proceddings of ICALP'98*, volume 1443 of *LNCS*, pages 103–115. Springer Verlag, 1998.

[16] 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.

[17] 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 G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations.* World Scientific, 1997.

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

[19] H.-J. Kreowski. A comparison between Petri nets and graph grammars. In H. Noltemeier, editor, *Proceedings of the Workshop on Graphtheoretic Concepts in Computer Science*, volume 100 of *LNCS*, pages 306–317. Springer Verlag, 1981.

[20] C. Lakos and S. Christensen. A general systematic approach to arc extensions for Coloured Petri Nets. In R. Valette, editor, *Proceddings of ICAPTN'94*, volume 815 of *LNCS*, pages 338–357. Springer Verlag, 1994.

[21] M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.

[22] M. Löwe, M. Korff, and A. Wagner. An Algebraic Framework for the Transformation of Attributed Graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, pages 185–199. Wiley, London, 1993.

[23] U. Montanari and F. Rossi. Contextual occurrence nets and concurrent constraint programming. In H.-J. Schneider and H. Ehrig, editors, *Proceedings of the Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*. Springer Verlag, 1994.

[24] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6):545–596, 1995.

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

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

[27] 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.