A Structural Approach to Graph Transformation Based on Symmetric Petri Nets

Lorenzo Capra*

Università degli Studi di Milano
Dipartimento di Informatica,Via Celoria 18, 20133 Milan, Italy

Abstract

Graph Transformation Systems (GTS) and Petri Nets (PN) are two central, theoretically sound, formal models for concurrent or distributed systems. A lot of papers have focused on the relationship between GTS and PN. It is generally accepted that PN are instances of GTS due to the lack of ability to adapt or reconfigure their structure. In this paper, which extends a recent one, we reverse this perspective by presenting a formal definition of GTS in terms of Symmetric Nets (SN). SN are a standard High-Level PN formalism featuring a compact syntax which highlights behavioural symmetries of systems. The major strength lies in the possibility of using well-established tools to edit/efficiently analyze (SN-based) graph transformation models. We follow a structural approach, based on a newly implemented symbolic calculus, to validate Graph Transformation Rules (GTR) and formally verify GTS properties.

In the second part of the paper, we supply a semantic characterization of the SN-based graph transformation model, by carrying out a thorough comparison with the classical, algebraic double-pushout (DPO) approach. We constructively and formally show how to translate any DPO rule to a corresponding, elementary or composite, SN rule. We treat both injective and non-injective DPO rules/rule matches. The comparison shows that the SN approach is, in some sense, a generalization of the DPO.

Keywords: Graph Transformation Systems, Double Pushout, Symmetric Nets, Structural analysis Operational semantics.

1. Introduction and Related Work

Graph transformation (or rewriting) [1, 2], introduced around fifty years ago [3], has been successfully proposed as a general, flexible, theoretically sound model for concurrent systems with a dynamical structure. Lots of research has focused on these topics. Many foundational papers describe different models of

*Corresponding author

graph transformation and make a comparison among them, using the category theory as a generalizing and unifying semantics [4, 5]. Many others present applications, especially in the area of networking, software engineering [2], and more recently, biology [6], social networks [7], chemistry [8].

The connections between Graph Transformation Systems (GTS) and other formalisms for concurrent systems, in particular, Petri Nets (PN) [9] and Process Algebra [10, 11], have been deeply investigated. GTS have borrowed some well known behavioural equivalence concepts from Process Algebra [12], which are similar in expressivity, but less flexible.

On the other side, PN are a central model for concurrent or distributed systems and represent a reference for any formalism meant to describe such systems, including GTS. The success of PN is due to several reasons, two of which particularly significant: a distributed state notion (corresponding to PN *marking*) that makes it easy to represent typical aspects of concurrency such as local conflicts, synchronizations, non-determinism, and the availability of a number of tools/techniques supporting the editing/analysis of models.

It is generally accepted that (classical) PN are instances of GTS, due to the lack of ability to adapt/reconfigure their structure. Some PN extensions have been thus equipped with dynamical reconfiguration capabilities (a survey may be found in [13]), which, however, almost always build on concepts typical of Graph Transformation Rules.

Kreowsky was the first to show that GTS are a generalization of some PN classes, in his seminal work [14] based on the double-pushout (DPO) approach. The idea is to represent a marked PN as a graph with three different types of nodes (for places, transitions, and tokens) and describe the firing of a PN transition through a rule (derivation). Since then, several encodings of PN classes in terms of GTS have appeared, among which Place/Transitions nets, Condition/Event nets, Elementary Net Systems, Consume-Produce-Read nets. Some PN variants with extra features, such as read/reset/inhibitor arcs, have also been encoded. It would be impossible to supply a complete list of these proposals, let us refer to [15] (and included references) for the earliest, and to [16, 17] for more recent ones.

This paper extends [18] and is somewhat related to most of those mentioned above. [18] considers the relationship between GTS and PN from a completely new perspective, by proposing a formalization of Graph Transformation Rules in terms of Symmetric Nets (SN) [19]. SN are a subclass of Coloured Petri nets [20, 21] featuring a particular syntax that outlines model symmetries and allows for efficient, both state-space based and structural, model analysis. The idea in [18] is to map a Graph Transformation Rule (GTR) to a SN transition connected to a couple of places whose state (marking) encodes a graph. [18] only deals with directed simple graphs.

Such an approach enjoys several benefits. You can exploit well-established tools for the editing/analysis of SN, like `GreatSPN` [22]. The Reachability Graph of a SN supplies the interleaving semantics of a GTS . The associated quotient graph (called Symbolic Reachability Graph) [23], which directly comes from a *symbolic* graph encoding, naturally abstracts from graph isomorphism. Above

all, some recent advances in SN symbolic structural analysis [24, 25], implemented in SNexpression tool [26] (`www.di.unito.it/~depierro/SNex`), allow you to efficiently validate rules and verify general properties of a GTS, such as conflicts, mutual-exclusion, semi-flows.

The first part of this paper significantly extends and consolidates the formalization of Graph Transformation Rules in terms of SN introduced in [18]. We consider a more general class of graphs, edge-labelled multi-graphs, and provide a sound structural characterization of SN rules by using the symbolic calculus implemented in `SNexpression`. We show the potential of structural analysis through simple but not trivial examples of property verification, including the use of semi-flows. We introduce a new kind of composite rule able to emulate graph transformations in undetermined contexts, which are peculiar to algebraic approaches to graph transformation.

In the second part, we conduct a thorough comparison of the SN based approach with a classical algebraic one, the double-pushout, or DPO [4]. We do not use the original, categorical pushout notions, but we follow an operational, though rigorous, schema. Starting from injective DPO rules, then considering non-injective ones, we show in a constructive, formal way how to translate any DPO rule to a corresponding, elementary or composite, SN rule. The intent is twofold: provide a sound, semantic characterization of the SN approach to graph transformation and, in a way, promote the interoperability among different formalisms/tools, a must in formal methods area. The comparison shows that the SN-based approach to graph transformation can be viewed as a generalization of (base) DPO. The possibility of encoding richer forms of graph transformations, such as Attributed Graph Rewriting, is part of our ongoing work. Let us finally remark that the use of the native stochastic extension of SN (SSN) would represent a natural encoding for Stochastic GTS.

All the examples presented in the paper are available in `GreatSPN` format at `github.com/lgcapra/GTS-SN`, together with a notebook including the commands to issue to the `SNexpression` shell for checking some of the structural formulae used in the examples.

The outline of the paper follows: Section 2 contains background notions and formally introduces SN; Sections 3 through 6 present the SN-based, structural approach to graph transformation: the encoding and formal validation of rules, the generation of a symbolic state-transition system which abstracts from graph isomorphism, some examples of property verification, the definition of composite rules; Section 7 formalizes the procedure(s) translating DPO rules into corresponding SN rules, starting from injective DPO rules/matches, then considering non-injective matches of a rule, lastly, non-injective rules; Section 8 draws some conclusion and outlines ongoing work.

2. Background

This section collects the background notions used in the rest of the paper, except for double-pushout (DPO) rules, introduced in Section 7.1. The reader

may refer to [9] and [19] for a detailed description of classical, low-level, PNs and SNs, respectively.

2.1. Multisets, multi-set functions, and their operations

A *multiset* (*bag*) over a domain D is a map $b : D \to \mathbb{N}$, where $b(d)$ is the *multiplicity* of d in b. The *support* $\overline{b}$ is $\{d \in D | b(d) > 0\}$: d is said an element of b ($d \in b$) if and only if $d \in \overline{b}$. A bag whose elements all have multiplicity one is said *type-set*. We may denote a multiset b as a weighted formal sum of its elements, weights representing multiplicities (omitted, if equal to one). With some overloading, a set symbol may also denote the corresponding type-set bag. The *null* multiset, i.e., the multiset with an empty support, is denoted ϵ_D, or just ϵ if its domain is implicit. $Bag[D]$ denotes the set of all bags over D.

Bag operations.. Let $b_1, b_2 \in Bag[D]$. The *sum* $b_1 + b_2$, the *difference* $b_1 - b_2$, and the *intersection* $b_1 \cap b_2$ are bags in $Bag[D]$ defined, for any $d \in D$, as: $b_1 + b_2(d) = b_1(d) + b_2(d)$; $b_1 - b_2(d) = b_1(d) - b_2(d)$ if $b_1(d) \geq b_2(d)$, 0 otherwise; $b_1 \cap b_2(d) = min(b_1(d), b_2(d))$. Two bags b_1, b_2 are said *disjoint* if $b_1 \cap b_2 = \epsilon$ ($\overline{b_1} \cap \overline{b_2} = \emptyset$). Associativity holds for $+, \cap$ (which are treated as n-ary operators), but not for $-$. Relational operators apply component-wise, e.g., $b_1 < b_2$ if and only if $\forall d \in D$, $b_1(d) < b_2(d)$. Similarly for the others.

The *scalar* product $k \cdot b_1$, $k \in \mathbb{N}$, is the bag $b'_1 \in Bag[D]$, s.t. for any $d \in D$ $b'_1(d) = k \cdot b_1(d)$. The hybrid notation $b + k$, $k \in \mathbb{N}$, stands for $b + k \cdot \overline{b}$, the bag obtained from b by increasing the elements' multiplicity by k (thus, $\epsilon + k = \epsilon$).

Let $b_i \in Bag[D_i]$, $d_i \in D_i$: the bag *Cartesian* product $b_1 \times b_2 \times \ldots b_\mathrm{n}$ belongs to $Bag[D_1 \times D_2 \times \ldots D_\mathrm{n}]$, and is defined as

$$b_1 \times b_2 \times \ldots b_\mathrm{n}(\langle d_1, d_2, \ldots, d_\mathrm{n} \rangle) = b_1(d_1) \cdot b_2(d_2) \cdot \ldots b_\mathrm{n}(d_\mathrm{n}).$$

Bag-functions. Bag-operators naturally extend to bag-functions. Let $f_1, f_2 : D \to Bag[D']$, and $op \in \{+, -, \cap\}$[1]: f_1 *op* $f_2 : D \to Bag[D']$ is defined f_1 *op* f_2 $(d) = f_1(d)$ *op* $f_2(d)$, $\forall d \in D$. Analogously, $\overline{f_1} : D \to 2^{D'}$ is defined $\overline{f_1}(d) = \overline{f_1(d)}$, $\forall d \in D$. As for relational operators, $f_1 < f_2$ if and only if $f_1(d) < f_2(d), \forall d \in D$. And so forth. The symbol $\epsilon_{D,D'}$ (just ϵ if the domains are implicit) denotes the constant null function of arity $D \to Bag[D']$. The notions of *type-set* bag and *disjoint* bags apply to bag-functions as well, considering function evaluation to all the arguments.

Let $f_i : D \to Bag[D_i]$. The scalar product $k \cdot f_i$ is defined as $k \cdot f_i(d), \forall d \in D$; the Cartesian product $f_1 \times f_2 \times \ldots f_\mathrm{n}$ is a map $D \to Bag[D_1 \times D_2 \times \ldots D_\mathrm{n}]$ such that $f_1 \times f_2 \times \ldots f_\mathrm{n}(d) = f_1(d) \times f_2(d) \times \ldots f_\mathrm{n}(d)$, $\forall d \in D$. The notation $\langle f_1, f_2, \ldots, f_\mathrm{n} \rangle$ (called *function-tuple*) will be used in place of $f_1 \times f_2 \times \ldots f_\mathrm{n}$.

Let $f : D \to Bag[D']$: the *transpose* $f^t : D' \to Bag[D]$ is defined as, $f^t(x)(y) = f(y)(x)$, $\forall x \in D', y \in D$. The function *linear extension* $f^* : Bag[D] \to Bag[D']$ is $f^*(b) = \sum_{x \in b} b(x) \cdot f(x)$, for any $b \in Bag[D]$. The

[1] we use, again, symbol overloading

linear extension applies to function composition as well: let $h: A \to Bag[B]$, $g: B \to Bag[C]$, then $g \circ h : A \to Bag[C]$ is defined as $g \circ h(a) = g^*(h(a))$. We shall henceforth use the same symbol for a bag-function and its linear extension.

Finally, let $\{f_i\}$ be a family of functions $f_i : D \to Bag[D']$. The *linear combination* $F = \sum_i \lambda_i \cdot f_i$, $\lambda_i \in \mathbb{Z}$, is a function $D \to Bag[D']$ if and only if $F(d)(d') = \sum_i \lambda_i \cdot f_i(d)(d') \geq 0$, $\forall d \in D, d' \in D'$.

2.2. Symmetric Nets

Symmetric Nets (SN)[2] [19] (`iso.org/standard/43538.html`) are a high-level PN standard formalism featuring a particular syntax which highlights behavioural symmetries of systems. SN show an acceptable trade-off between expressivity and analysis capability. Efficient algorithms are available for both state-space based [23] and *structural* [24, 25] analysis of SN. Many of these are integrated in `GreatSPN` [22], a graphical editor for SN, whereas the most recent advances on structural analysis are implemented in `SNexpression` [26] (`www.di.unito.it/~depierro/SNex`), a kind of computer-algebra system.

SN are a particular flavour of *Colored* Petri Nets (CPN) [21]. Like in any PN formalism, the underlying structure is a finite, directed bipartite graph, whose nodes are partitioned in $P \cup T$, P and T being non-empty sets, holding the *places* and *transitions*, respectively. The former, drawn as circles, represent state variables, whereas the latter, events causing local state changes.

The SN formalism admits two kinds of transitions: those drawn as white rectangles represent *observable* (time-consuming, in Stochastic SN) events, whereas those drawn as tiny black bars, called *immediate*, model *non-observable* (logical) events. The latter take priority over the former. As in any high-level PN formalism, (colour) domains are associated with SN nodes. Each edge (called arc in SN) is annotated by a function that, given an element of the (edge's) transition's domain, yields a bag defined on the place's domain. In the next sections, we formally introduce the SN color annotations and semantics.

2.2.1. SN Color Domains

The color structure of a SN model is built upon the *basic color classes* $\mathcal{C} = \{C_i, i = 1 \ldots, \mathrm{n}\}$, finite, pair-wise disjoint sets, which may be (circularly) *ordered* or *partitioned* into *static subclasses* $C_{i,j}$. A *color domain* $\mathcal{D}$ is a Cartesian product of classes : $\mathcal{D} = \prod_{i=1,\ldots,\mathrm{n}} C_i^{e_i}$, $e_i \in \mathbb{N}$ being the repetitions of class C_i in $\mathcal{D}$. The color domain of place p, $\mathcal{D}(p)$, defines the type of *tokens* (color-tuples) p may hold; the color domain of transition t, $\mathcal{D}(t)$, instead defines the potential firing *instances* of t: we use typed local variables, $Var(t)$, to refer to single elements in a color-tuple of $\mathcal{D}(t)$. Formally, transition variables are *projections*, as explained below.

The partition of color classes into static subclasses is what determines the symmetry level of an SN model : each static subclass gathers (all and only) system entities of a given type equivalently behaving, therefore, indistinguishable.

[2] Originally introduced with the name of Well-formed Nets, in their stochastic extension.

We shall use simple conventions. Capital letters, e.g. C, will denote color classes. A capital letter with a subscript, e.g., C_i, will denote a subclass of a given class.

The SN models used in the paper build on two *unordered* color classes: N=$\{v_i, i : 1 \ldots |\mathrm{N}|\}$ and $\mathrm{L} = \bigcup_{j:1\ldots|\mathrm{L}|} \mathrm{L}_j$, with $\mathrm{L}_j = \{lb_j\}$, $\forall j$. Color domains take the form: $\mathrm{N}^{e_1} \times \mathrm{L}^{e_2}$.

A transition variable will be denoted (with a few exceptions) by a single, (possibly) subscripted lower-case letter, e.g. n_1, which implicitly indicates the variable's type (N). Subscripts distinguish variables of the same class if it repeatedly occurs in a transition's domain. Otherwise, the subscript is optional. As an example, the color domain of transition t_{r_1} (Figure 1 (R1)) is $\mathrm{N}^3 \times \mathrm{L}$. The transition's variables are $\{n_i\}$, $i : 1 \ldots 3$, and l.

SN nodes may have a *neutral* color domain consisting of a singleton color class X. In particular, if $Var(t) = \emptyset$ then $\mathcal{D}(t) = \mathrm{X}$.

2.2.2. Guards and Transition Instances

A transition t may have a *guard*, $g(t)$, a boolean expression on $\mathcal{D}$(t) built of *basic predicates*, such that $Var(g(t)) \subseteq Var(t)$. The only used in this paper are (we refer to a generic class C):

- $c_1 = (\neq) c_2$, true when c_1 and c_2 are assigned the same/a different color
- $c_1 \in \mathrm{C}_j$, true when the color assigned to c_1 belongs to subclass C_j;

The default guard is the constant *true*, and is omitted.

A *transition instance* is denoted (t, b), $b \in \mathcal{D}(t)$. It may be seen as an assignment (binding) of colors to $Var(t)$. For example, an instance of transition t_{r_1} (Figure 1 (R1)) is $b :\ (n_1 = v_2,\ n_2 = v_1,\ n_3 = v_3,\ l = lb_1)$. We say that (t, b) is *valid* if and only if $g(t)(b) = true$. With *transition color domain* we shall henceforth refer to the restriction of $\mathcal{D}$(t) to valid instances of t.

2.2.3. Arc Functions

An arc is said *input* if going from a place to a transition, *output* if going in the opposite direction. In SN there is another type of arc, called *inhibitor*, drawn with an ending small circle on the transition. SN arcs are annotated by families of functions, denoted (for each type) I$[p, t]$, O$[p, t]$, H$[p, t]$, respectively. Formally, an *arc-function* $F[p, t] :\ \mathcal{D}(t) \rightarrow Bag[\mathcal{D}(p)]$ is expressed as a linear combination:

$$F[p, t] = \sum_k \lambda_k . T_k[g_k],\ \lambda_k \in \mathbb{Z}, \tag{1}$$

where T_k is a tuple (Cartesian product) $\langle f_1, \ldots, f_{\mathrm{h}} \rangle$ of *class-functions*, possibly suffixed by a guard on $\mathcal{D}(t)$: if $g_k(b) = true$, then $T_k[g_k](b) = f_1(b) \times \ldots f_{\mathrm{h}}(b)$, otherwise, $T_k[g_k](b) = \epsilon$. Scalars in (1) must ensure that the linear combination is a bag-function. If $\mathcal{D}(p) = \mathrm{X}$ then (for simplicity) $F[p, t] \in \mathbb{N}$.

A class-C function f_i is a *type-set* map $\mathcal{D}(t) \rightarrow Bag[\mathrm{C}]$, defined in terms of *elementary functions* $\mathcal{E}_\mathrm{C} = \{c_j, {+}{+}c_j, {-}{-}c_j, \mathrm{C}_q, All\}$:

- c_j (variable, or *projection*) maps a color-tuple $b \in \mathcal{D}(t)$ to the j^{th} occurrence of color C in b; if C is ordered, $++c_j$ ($--c_j$) yields the $\mathrm{mod}_{|\mathrm{C}|}$ successor (predecessor) of the color bound to c_j[3];
- C_q (defined only if C is partitioned) and *All* are *constant* maps to C_q and C, respectively.

Formally, a class-function f_i is recursively defined:

$$f_i = \begin{cases} e,\ e \in \mathcal{E}_{\mathrm{C}} \\ e \pm f_i,\ e \in \mathcal{E}_{\mathrm{C}} \end{cases} \quad \text{or} \tag{2}$$

where in (2) $\pm$ are meant as *set-operations*. This syntax/semantics is fully in accordance with that of `GreatSPN` [22] and `SNExpression` [26] tools, whereas is slightly different (even if equivalent) from that used in legacy papers on SNs, where class-functions are defined in turn as linear combinations. We shall exploit it when handling non-injective graph morphisms.

As an example, consider the function-tuple $\langle n_1, n_2 + n_3, \mathrm{L}_1 \rangle : \mathrm{N}^3 \to \mathrm{N}^2 \times \mathrm{L}$, which annotates the I/O arcs of transition t_r in Figure 12. When evaluated on a color-tuple $\langle v_1, v_2, v_3 \rangle$, it yields the type-set bag $1 \cdot \langle v_1, v_2, lb_1 \rangle + 1 \cdot \langle v_1, v_3, lb_1 \rangle$, instead, when evaluated on a color-tuple $\langle v_1, v_2, v_2 \rangle$ yields $1 \cdot \langle v_1, v_2, lb_1 \rangle$. Its semantics is thus different from the sum $\langle n_1, n_2, \mathrm{L}_1 \rangle + \langle n_1, n_3, \mathrm{L}_1 \rangle$, which, when evaluated on $\langle v_1, v_2, v_2 \rangle$, results in $2 \cdot \langle v_1, v_2, lb_1 \rangle$.

It may be convenient/necessary to rewrite arc functions as weighted sums of pair-wise disjoint terms, according to:

Property 1 Any SN arc-function F can be equivalently expressed as F'

$$F' = \textstyle\sum_i \lambda_i T_i[g_i],\ \lambda_i \in \mathbb{N},\ \text{where } \forall i, j, i \neq j : T_i[g_i] \cap T_j[g_j] = \emptyset.$$

For example: $2 \cdot \langle All - n_1, n_2 \rangle [n_1 \neq n_2] + 1 \cdot \langle n_2, n_2 \rangle \ : \ \mathrm{N^k} \to \mathrm{N}^2,\ \mathrm{k} > 1$
$\equiv 2 \cdot \langle All - n_1 - n_2, n_2 \rangle [n_1 \neq n_2] + 3 \cdot \langle n_2, n_2 \rangle [n_1 \neq n_2] + 1 \cdot \langle n_2, n_2 \rangle [n_1 = n_2]$.

2.2.4. Semantics of SN

A *marking* $\mathbf{m}$ provides a distributed notion of state. Formally, $\mathbf{m}$ is a P-indexed vector such that $\mathbf{m}[p] \in Bag[\mathcal{D}(p)]$. The elements of $\mathbf{m}[p]$ (the marking of place p) are said *tokens*. If p is neutral then (for simplicity) $\mathbf{m}[p] \in \mathbb{N}$.

The *firing rule* sets the SN interleaving semantics. We assume that missing arcs are annotated by null functions. A transition instance (t, b) *has concession* in marking $\mathbf{m}$ if and only if:

- $\forall p \in P$: $\mathrm{I}[p, t](b) \leq \mathbf{m}[p]$
- $\forall p \in P\ \forall x \in \mathrm{H}[p, t](b)$: $\mathrm{H}[p, t](b)(x) > \mathbf{m}[p](x)$

[3] in this paper we only use unordered classes

An instance (t, b) having concession in $\mathbf{m}$ is *enabled* if no higher-priority transition instance has concession in $\mathbf{m}$. In that case, (t, b) may *fire*, withdrawing the bag $\mathrm{I}[p, t](b)$ from each (input) place p and putting the bag $\mathrm{O}[p, t](b)$ into each (output) place p. The reached marking, $\mathbf{m}'$, is defined as:

$$\mathbf{m}'[p] = \mathbf{m}[p] - \mathrm{I}[p, t](b) + \mathrm{O}[p, t](b), \ \forall p \in P$$

We say that $\mathbf{m}'$ is *reachable* from $\mathbf{m}$ through (t, b), denoted $\mathbf{m}[t, b\rangle\mathbf{m}'$.

We call a marking $\mathbf{m}$ *vanishing* if and only if there is some immediate transition instance which is enabled in $\mathbf{m}$, *tangible* otherwise. If we set a *tangible initial marking* $\mathbf{m}_0$, and there are no infinite firing sequences of immediate transition instances, we can build the *reachability graph* (RG) of an SN model.

The RG is an edge-labelled, directed multi-graph (N_{RG}, E_{RG}) (this structure will be formalized next) whose nodes are tangible markings, defined as follows: $\mathbf{m}_0 \in N_{RG}$; if $\mathbf{m} \in N_{RG}$, and $\mathbf{m}[t, b\rangle\mathbf{m}_1[t_1, b_1\rangle \dots \mathbf{m}_n[t_n, b_n\rangle\mathbf{m}'$, where $\mathbf{m}'$ is a tangible marking and $\{(t_i, b_i), i : 1 \dots n\}$, $n \geq 0$, is a (*possibly empty*) sequence of *immediate* transition instances, then $\mathbf{m}' \in N_{RG}$ and $\mathbf{m} \xrightarrow{t,b} \mathbf{m}' \in E_{RG}$.

2.3. Graphs and Graph Morphisms

We choose to consider a quite general class of graphs, used in many papers, namely directed, edge-labelled, multi-graphs (parallel edges are allowed). Other possible choices, such as the use of hypergraphs or attributed graphs, will be shortly discussed in section 8. Let Λ be a fixed set of labels.

Graph. A graph G is a 5-tuple $G = (N, E, s, t, l)$ where

- N is a set of nodes, E is a set of edges, $N \cap E = \emptyset$
- $s : E \to N$ is the *source* function; $t : E \to N$ is the *target* function
- $l : E \to \Lambda$ is the *labelling* function

The components of a graph will be subscripted by the graph's name, if needed. The source and target nodes of an edge $e \in E$ are said *incident* to e. A crucial notion is that of (total) graph morphism. Let G, H be graphs.

Graph Morphism. A graph morphism $\phi : G \to H$ is a pair of functions $\phi_N : N_G \to N_H$, $\phi_E : E_G \to E_H$, such that $\forall e \in E_G$:

$$s_H(\phi_E(e)) = \phi_N(s_G(e)),\ t_H(\phi_E(e)) = \phi_N(t_G(e)),\ l_H(\phi_E(e)) = l_G(e)$$

A morphism maps the nodes and the edges of a graph to the nodes and the edges of another, by preserving graph structure and edge labels. A morphism $\phi : G \to H$ is said *injective* (*surjective*) if both ϕ_N and ϕ_E are. If ϕ is both injective and surjective ϕ is said an *isomorphism*, written $G \cong H$. The set of morphisms between multi-graphs $G \to H$ has an associated, obvious equivalence relation: ϕ_1, ϕ_2 are equivalent if they coincide, up to the mapping of parallel edges. We implicitly consider the representatives of equivalence classes.

Graph morphisms are base components of graph transformation rules and are used to match the left-hand side of a rule to a host graph.

Another key concept is that of graph *gluing*, usually called *pushout* in the categorical setting. The term gluing has an intuitive explanation: assume that G and H are two graphs with a common interface (i.e., an overlap) I, then gluing G and H through I, results in a graph, denoted $G +_I H$, obtained by "joining" G and H via their overlap I. The embedding of I into G and H is formally described by two morphism $\phi_H : I \to H$, $\phi_G : I \to G$. The intuition above is valid if both morphisms are injective, whereas is partially incorrect if they are not: in such a case, in fact, there may be some merge of graph elements, formally captured by definition below using an equivalence class.

Given a set A, and an equivalence relation $\equiv$ on A, the set of equivalence classes is denoted $A/\equiv$, whereas the equivalence class of $a \in A$ is denoted $[a]_\equiv$. In all subsequent definitions, we assume that node/edge sets are disjoint.

Graph Gluing. Let I, G, H, be graphs, and $\phi_H : I \to H$, $\phi_G : I \to G$ be graph morphism (I is called *interface*). Let $\equiv$ be the smallest equivalence relation on $E_G \cup N_G \cup E_H \cup N_H$ such that $\forall x \in N_I \cup E_I \ \phi_H(x) = \phi_G(x)$. The gluing of G, H, over I, (denoted $G +_I H$ or $G +_{\phi_H,\phi_G} H$) is a graph X such that

$$N_X = (N_G \cup N_H)/\equiv \quad E_X = (E_G \cup E_H)/\equiv$$

$$s_X([e]_\equiv) = \begin{cases} [s_G(e)]_\equiv \text{ if } e \in E_G \\ [s_H(e)]_\equiv \text{ if } e \in E_H \end{cases} \qquad t_X([e]_\equiv) = \begin{cases} [t_G(e)]_\equiv \text{ if } e \in E_G \\ [t_H(e)]_\equiv \text{ if } e \in E_H \end{cases}$$

$$l_X([e]_\equiv) = \begin{cases} l_G(e) \text{ if } e \in E_G \\ l_H(e) \text{ if } e \in E_H \end{cases}$$

Intuitively, we obtain $G +_I H$ by juxtaposing the parts of G and H that do not belong to the images of ϕ_H, ϕ_G, and by merging the elements of G and H with a common pre-image.

3. Encoding Graph Transformation Rules in SN

In this section, we formally introduce the graph transformation model based on SN, focusing on graph encoding and structural conditions featuring well-defined rules. We consider by now *elementary* rules encoded by single transitions. The benefits of this approach are basically three: the possibility of using a symbolic structural calculus for validating single rules and verifying properties of GTS; the (implicit) abstraction from graph isomorphisms when using a symbolic encoding of graphs; the availability of well-established tools supporting SN, like `GreatSpn` and `SNExpression`. We shall use a few examples to illustrate the base concepts.

Even if we refer to directed (edge labelled) multi-graphs, we may easily extend this approach to other classes of graphs, e.g. multipartite graphs.

3.1. Graph Encoding as an SN Marking

The basic idea is to encode a graph through a pair of SN places, `Node` and `Edge`, which are then suitably connected to any transition (subnet) representing a rule.

Two basic color classes N, L are used: $\mathrm{N} = \{v_i, i : 1 \ldots |\mathrm{N}|\}$ holds node descriptors, and is assumed large enough to cover all possible evolutions of a graph; $\mathrm{L} = \mathrm{L}_1 \cup \mathrm{L}_2 \cup \ldots \mathrm{L}_m$ holds label descriptors, i.e., $\mathrm{L}_i = \{lb_i\}$. Their color domains are: $\mathcal{D}(\texttt{Node}) = \mathrm{N}$, $\mathcal{D}(\texttt{Edge}) = \mathrm{N}^2 \times \mathrm{L}$.

A graph $G = (N_G, E_G, s, t, l)$ is encoded as an SN marking, denoted $\mathbf{m}_G$. Whenever possible, we use the same symbols for graph nodes, edge labels, and corresponding colors of classes N and L, respectively. We assume $|N_G| \leq |\mathrm{N}|$ and $|\Lambda| = |\mathrm{L}|$[4].

$$\mathbf{m}_G[\texttt{Node}] = \sum_{v_i \in N_G} v_i \qquad \mathbf{m}_G[\texttt{Edge}] = \sum_{e \in E_G} \langle s(e), t(e), l(e) \rangle$$

.

Observe that $\mathbf{m}_G[\texttt{Node}]$ is a *type-set* bag, whereas the elements of $\mathbf{m}_G[\texttt{Edge}]$ may have a multiplicity greater than one (denoting parallel edges).

In general, the class-N colors describing the nodes of graph G in $\mathbf{m}_G$ are denoted by an injective map $col_G : N_G \to \mathrm{N}$.

Vice-versa, the following definition sets the conditions under which an SN marking encodes a graph:

Definition 1 (Graph Encoding) Marking $\mathbf{m}$ is a *graph-encoding* if and only if

- $\mathbf{m}[\texttt{Node}]$ is type-set
- $\langle n_1 + n_2 \rangle \circ \mathbf{m}[\texttt{Edge}] \leq \mathbf{m}[\texttt{Node}]$

The 2nd condition in Definition 1 means that there are no *dangling edges*, i.e., edges with non-encoded, incident nodes. The graph corresponding to a graph-encoding $\mathbf{m}$ is denoted $G_{\mathbf{m}}$ (we skip its formalization, as trivial).

The adopted graph-encoding is not the only possible. A compact alternative might just use place `Edge`, and the extra subclass $\mathrm{L}_0 = \{lb_0\}$: isolated nodes would be represented by color tuples $\langle v_i, v_i, lb_0 \rangle$. This choice, however, would require to check for new (non-)isolated nodes after any edge deletion (insertion).

3.2. SN Graph Transformation Rules: a Structural Approach

Graph Transformation Rules (GTR) are formalized as SN subnets including places `Node` and `Edge`. We first consider *elementary* rules consisting of single observable transitions linked to these places (shared among the rules of a GTS), then we introduce and shortly discuss on *composite* rules, described by more complex subnets.

[4]We may also write $N_G \subseteq \mathrm{N}$, $\Lambda \cong L$

An *elementary* GTR is implemented by a SN transition t_r, whose color domain is implicitly defined by $Var(t_r)$: in general, $\mathcal{D}(t_r) = \mathrm{N}^{e_1} \times \mathrm{L}^{e_2}$, $e_1 > 0$.

The idea is simple: the input arc functions $\mathrm{I}[\texttt{Node}, t_r]$, $\mathrm{I}[\texttt{Edge}, t_r]$ (which should not be both null), and the inhibitor arc function $\mathrm{H}[\texttt{Edge}, t_r]$, when evaluated on an enabled instance of t_r in a graph-encoding $\mathbf{m}$, *match* a subgraph of the encoded graph, which is rewritten according to the SN firing rule: instantaneously, the matched subgraph is removed from the encoded graph and replaced with another subgraph yielded by evaluating the output arc functions on the same instance. The role of inhibitor arc-functions is to set the rule's application context (e.g., a node without predecessors) and ensure general conditions of correctness (e.g., the dangling edge condition).

Figure 1 is a gallery of rule representatives. R1 (when repeatedly applied) builds the (label-preserving) transitive closure of a graph: it requires an initial simple graph to properly work. R2 removes isolated nodes from a graph. R3 derives a Kripke structure from a source graph, by creating an edge-loop with label lb_1 for nodes without successors. R4 removes parallel edges, if any. R5 deletes a node with only one incident edge-loop: the quite complex inhibitor arc function ensures that there are no dangling edges. Finally, R6 is matched by a couple of edges with source v_i and target different from v_i, with the same label: these two edges are replaced by parallel edges with source v_i and target a fresh new node (bound to n_4), with label lb_1.

3.3. Elementary Graph Transformation Rules

A transition t represents a Graph Transformation Rule if *any* instance of t transforms a graph-encoding into another one, as formalized by:

Definition 2 (Graph Transformation Rule, GTR) A SN transition t is a GTR if and only if, for any graph-encoding $\mathbf{m}$, $\forall b \in \mathcal{D}(t)$:

$$\mathbf{m}[t, b\rangle\mathbf{m}' \Rightarrow \mathbf{m}' \text{ is a graph-encoding.}$$

Definition 2 refers to any graph/transition instance. Therefore, we need for parametric *structural* conditions ensuring that an SN transition represents a GTR. By exploiting the SN calculus introduced in [25, 24] and recently implemented in `SNexpression` tool [26], we can mechanically derive these conditions from the arc-functions surrounding a transition and efficiently manipulate the corresponding symbolic expressions.

`SNexpression` is a kind of symbolic calculator, which allows checking structural properties (conflict, causal connection, mutual exclusion, semiflows, and many others lower-level) directly on an SN, without any unfolding. The SN calculus solves algebraically any formulae defined in terms of a language, $\mathcal{L}$, and a base set of functional operators (difference, intersection, composition, transpose, and support). The syntax of $\mathcal{L}$ is a small extension of arc-functions' syntax, in which guards defined on functions' co-domain may *prefix* functions, performing as *filters*: letting $f : A \to Bag[B]$, then $[g]f : A \to Bag[B]$ is such that if

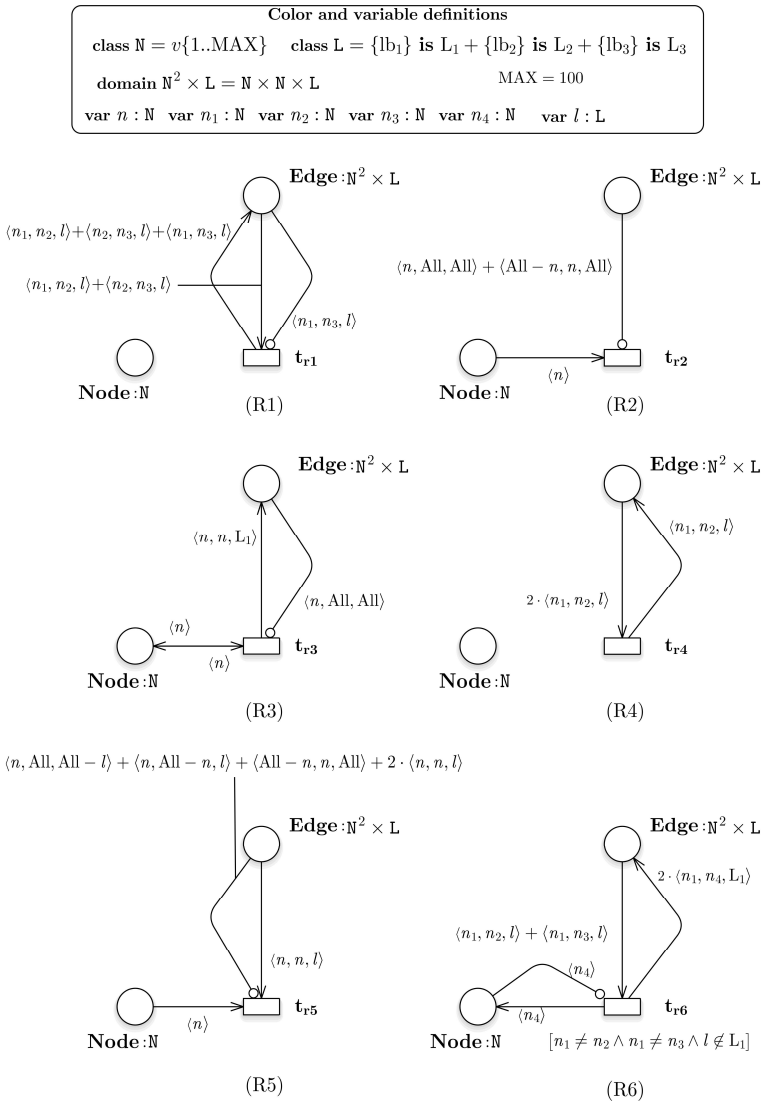


Figure 1: Examples of elementary Graph Transformation Rules

Table 1: Symbolic expressions used in SN GTR structural definition.

Symbol	Definition/Description
$\mathcal{D}(t) \rightarrow Bag[\mathcal{D}(p)]$	
$\mathrm{W}^+[p,t]$	$\mathrm{O}[p,t] - \mathrm{I}[p,t]$ multi-set of colors put in place p by an instance of t
$\mathrm{W}^-[p,t]$	$\mathrm{I}[p,t] - \mathrm{O}[p,t]$ multi-set of colors removed from p by an instance of t
$\mathcal{D}(t) \rightarrow Bag[\mathcal{D}(\mathtt{Edge})]$	
F^*	$\langle \mathrm{W}^-[\mathtt{Node},t], All, All\rangle + \langle All - \mathrm{W}^-[\mathtt{Node},t], \mathrm{W}^-[\mathtt{Node},t], All\rangle$ "set" of edges incident to any node removed by an instance of t
$\mathrm{W}^-[\mathtt{Edge},t]^*$	$\sum_i \lambda_i \cdot F_i \cap F^*$, letting $\mathrm{W}^-[\mathtt{Edge},t] = \sum_i \lambda_i \cdot F_i$ multi-set of edges removed by an instance of t which are incident to withdrawn nodes

$g(x) = true$ then $[g]f(y)(x) = f(y)(x)$, otherwise $[g]f(y)(x) = \epsilon$, $\forall y \in A, x \in B$. `SNexpression` works as a rewriting system by reducing any expression e, possibly involving the operators mentioned above, to a *normal form* $e' \in \mathcal{L}$, a kind of disjunctive normal form in which the only admitted operators are '+' and (at class-function level) '$\cap$'. We denote it: $e \rightarrow e'$.

`SNexpression` manages both bag-functions and their supports. In the sequel, we shall use the overloaded operator symbols, '$-$,$+$,$\cap$', denoting either bag- or set-operators, depending on their operands. We can syntactically check the equivalence between expressions, thanks to the following property:

$$e \equiv \epsilon\ (\emptyset) \ \Leftrightarrow e \rightarrow \epsilon\ (\emptyset)$$

In particular, we shall implicitly use two intuitive equivalences. Let F, F' be bag-functions with the same (co-)domains:

$$F \leq F' \Leftrightarrow F - F' \equiv \epsilon \qquad \overline{F} \subseteq \overline{F'} \Leftrightarrow \overline{F} - \overline{F'} \equiv \emptyset.$$

Lemma 1 states quite general conditions for a SN transition t to represent a GTR. The first one is related to the assumption that the marking of place `Node` is type-set, whereas the other two, a bit more complex, concern dangling-edges. We use some auxiliary expressions, defined and described in Table 1.

Lemma 1 *Let t be a SN transition linked to places* `Node`, `Edge`. *If the following conditions hold:*

C 1. $\mathrm{H}[\mathtt{Node},t] \leq \langle All\rangle \ \wedge\ \mathrm{W}^+[\mathtt{Node},t] \leq \mathrm{H}[\mathtt{Node},t]$

C 2. $\langle n_1 + n_2\rangle \circ \mathrm{O}[\mathtt{Edge},t] - \langle n_1 + n_2\rangle \circ \mathrm{I}[\mathtt{Edge},t] \leq \mathrm{O}[\mathtt{Node},t]$

C 3. $\mathrm{H}[\mathtt{Edge},t] \equiv H_1 + H_2,\ H_1 \cap H_2 \equiv \epsilon$
with $H_1 = (F^* - \mathrm{W}^-[\mathtt{Edge},t]) + (\mathrm{W}^-[\mathtt{Edge},t]^* + 1)$

then t is a GTR.

PROOF. Let $\mathbf{m}$ be a graph-encoding, $b \in \mathcal{D}(t)$, and $\mathbf{m}[t,b\rangle\mathbf{m}'$. The 1st inequality in Condition 1 ensures that $\mathrm{H}[\mathtt{Node},t](b)$ is a type-set bag: due to the enabling condition, it means that no element of this bag and (for the 2nd inequality) of the bag added to place `Node`, which is in turn type-set, must be present in $\mathbf{m}[\mathtt{Node}]$. Consequently, $\mathbf{m}'[\mathtt{Node}]$ is type-set. Condition 2) avoids the creation of dangling edges due to *insertion of edges*: $\mathrm{O}[\mathtt{Edge},t](b)$ is the bag of newly added edges, the minuend consequently is the set of nodes to which these edges are incident. By subtracting the nodes to which existing edges are incident (which are encoded, due to Definition 1) we get the fresh nodes involved in $\mathrm{W}^+[\mathtt{Edge},t](b)$, enforcing that they are contextually put in place `Node`. Condition 3) avoids the creation of dangling edges due to *node removal*. First, observe that $\mathrm{W}^-[\mathtt{Node},t](b)$, the bag of withdrawn nodes, is actually type-set, otherwise (t,b) would not be enabled in $\mathbf{m}$. The inhibitor arc function ensures that, for every withdrawn node, there is no edge incident to it, but for those contextually removed. The term $t_1 = (F^* - \mathrm{W}^-[\mathtt{Edge},t])(b)$ is (by definition) a type-set symbolic bag representing all and only the edges incident to any removed node, with the exclusion of those which are contextually withdrawn; $t_2 = (\mathrm{W}^-[\mathtt{Edge},t]^* + 1)(b)$ is a multi-set including all and only the withdrawn edges which are incident to any withdrawn node, with their multiplicity increased by one. The two terms above are disjoint by construction. Since the inhibitor arc function $\mathrm{H}[\mathtt{Edge},t]$ includes both terms, plus (possibly) another one disjoint as well, and (t,b) is enabled, we have: if $x \in t_1$, i.e., x is an edge incident to a removed node, which is not contextually removed, then x is not encoded; otherwise, if $x \in t_2$, i.e., x is an edge incident to a removed node, and x is contextually removed, then it is encoded in exactly as many instances as those withdrawn (formally, it holds $\mathrm{I}[\mathtt{Edge},t](b)(x) \leq \mathbf{m}[\mathtt{Edge}](b)(x)$ and $\mathrm{H}[\mathtt{Edge},t](b)(x) = \mathrm{W}^-[\mathtt{Edge},t](b)(x) + 1 > \mathbf{m}[\mathtt{Edge}](b)(x)$, thus, since $\mathrm{W}^-[p,t] \leq \mathrm{I}[p,t]$, we get: $\mathrm{I}[\mathtt{Edge},t](b)(x) = \mathbf{m}[\mathtt{Edge}](b)(x)$). ■:

Let us point out two relevant aspects of Lemma 1.

1. No constraint is set on $\mathrm{I}[\mathtt{Node},t]$ because Definition 2 only concerns *enabled* instances. Since we assume that the marking of place `Node` is type-set, every instance (t,b) such that $\mathrm{I}[\mathtt{Node},t](b)$ is not type-set is excluded. Thus, there may be implicit guards: for example, if $\mathrm{I}[\mathtt{Node},t] = \langle n_1\rangle + \langle n_2\rangle$, it is implicit $n_1 \neq n_2$, if $\mathrm{I}[\mathtt{Node},t] = \langle All - n_1\rangle + \langle n_2\rangle$, it is implicit $n_1 = n_2$.
2. Condition 3) explicitly shows what $\mathrm{H}[\mathtt{Edge},t]$ looks like: this makes it possible to mechanically derive it from $\mathrm{W}^-[\mathtt{Node},t]$ and $\mathrm{W}^-[\mathtt{Edge},t]$.

We have successfully verified the conditions set by Lemma 1 on all transitions shown in Figure 1, by means of `SNexpression`. The check, basically a term-equivalence test, took (at most) a few dozen ms for each involved calculation.

Structured Labels. In our encoding of graphs edge labels are coherently represented as elementary objects, corresponding to singleton subclasses of L. To enhance the model's expressivity, we might, however, use richer structured labelling of edges, and retain all the theoretical achievements. A label could be

expressed as a color-tuple of whatever color domain, on which all available SN functions could apply. This goes in the direction of those graph rewriting extensions introduced for modelling reasons, like Attributed Graph Rewriting [27, 28]. In particular, the characterization of SN Graph Transformation Rules provided by Lemma 1 would still work, but for inserting as many occurrences of the *All* constant function as the label domain's size, at the end of function-tuples. Also, all the outcomes of the next sections would apply with a few adaptations.

3.4. Graph Transformation System

An SN Graph Transformation System consists of an initial graph and a set of SN graph transformation rules.

Definition 3 (SN Graph Transformation System) Let G_0 be a graph, and $\mathcal{R}$ be a set of SN Graph Transformation Rules (Definition 2). The corresponding GTS is obtained by sharing places `Node` and `Edge` among $\mathcal{R}$ transitions, and setting $\mathbf{m}_{G_0}$ as initial marking.
The GTS state-transition system corresponds to the SN Reachability Graph.

Consider, as an example, the SN in Figure 2 bringing together the elementary rules R1,R3 of Figure 1. Given a graph G_0 encoded by the initial marling $\mathbf{m}_{G_0}$, the SN Reachability Graph describes the sequence of transformations that G_0 undergoes by applying either Rule 1 or Rule 3. The resulting RG has an *absorbing* state, i.e. a dead home-state, which corresponds to the transitive closure of G_0, where, also, nodes without proper predecessors have newly added, incident edge-loops.

Let $\mathbf{m}_{G_0}[\texttt{Edge}] = \langle v_1, v_2, lb_1\rangle + \langle v_1, v_3, lb_1\rangle + \langle v_4, v_1, lb_1\rangle$, and $\mathbf{m}_{G_0}[\texttt{Node}] = \langle v_1 + v_2 + v_3 + v_4\rangle$: the corresponding RG (built with the `GreatSPN` package) holds 16 nodes, including the absorbing one:

$$\langle v_1, v_2, lb_1\rangle + \langle v_1, v_3, lb_1\rangle + \langle v_4, v_1, lb_1\rangle + \langle v_2, v_2, lb_1\rangle + \langle v_3, v_3, lb_1\rangle + \langle v_4, v_2, lb_1\rangle + \langle v_4, v_3, lb_1\rangle$$

.

4. Graph Isomorphism Abstraction and SN Symbolic Marking

In this section, we introduce the SN Symbolic Marking notion (with some related concepts) and discuss the isomorphism abstraction it implicitly provides in the graph encoding context.

In the context of GTS, all considerations are valid up to graph isomorphism. In particular, the state-transition system associated with a GTS usually abstracts from isomorphic graphs obtained by applying rules.

The system symmetries implicitly expressed by the syntax of the SN formalism, in the graph encoding context, exactly match graph isomorphism. Let us recall a few basic concepts and refer to [23] for the details.

SN are equipped with a syntactical state-equivalence notion, called *symbolic marking* (SM). Two SN markings $\mathbf{m}_1$, $\mathbf{m}_2$ belong to the same SM $\widehat{\mathbf{m}}$ if and only if there is a permutation σ on color classes (a rotation, on ordered classes),

preserving static subclasses, such that $\mathbf{m}_2 = \sigma(\mathbf{m}_1)$ (we write $\mathbf{m}_1 \equiv \mathbf{m}_1$). An immediate consequence is the following.

Property 2 $G \cong G'$ if and only if there exists a permutation σ of class-N colors such that $\mathbf{m}_{G'} = \sigma(\mathbf{m}_G)$.

Note that the only permutation on L is the identity.

We might easily extend this intuitive property if edge labels had a richer structure, as discussed later.

At transition instance level, the SN symmetry looks like a *strong bisimulation*: for each color permutation σ: $\mathbf{m}[t,b\rangle\mathbf{m}' \Leftrightarrow \sigma(\mathbf{m})[t,\sigma(b)\rangle\sigma(\mathbf{m})'$; on the other side, $\mathbf{m}_1[t,b\rangle\mathbf{m}_2 \Rightarrow \forall\mathbf{m}'_2,\mathbf{m}'_2 \equiv \mathbf{m}_2\ \exists\mathbf{m}'_1,b',\mathbf{m}'_1 \equiv \mathbf{m}_1 \wedge b' \equiv b$: $\mathbf{m}'_1[t,b'\rangle\mathbf{m}'_2$.

By setting an *initial symbolic marking*, $\widehat{\mathbf{m}}_0$, we can thus automatically generate (with the `GreatSPN` package) a quotient graph, called Symbolic Reachability Graph (SRG), which retains all the liveness/safety properties of the ordinary RG of a SN. Formally, a symbolic marking (SM) employs *dynamic subclasses* instead of colours. Dynamic subclasses define (locally) a *parametric partition* of color (sub-)classes: each dynamic subclass refers to a specific static subclass, or a colour class if that class is not partitioned. The *size* of a dynamic subclass indicates a set of different colors evenly distributed over the SN places. Consider, for example, the initial symbolic marking below, where symbol zv_i denotes a class-N dynamic subclass, whereas static subclasses denote edge labels:

$$\widehat{\mathbf{m}}_0[\texttt{Edge}] = \langle zv_1, zv_2, \mathrm{L}_1\rangle + \langle zv_3, zv_1, \mathrm{L}_1\rangle,\ \widehat{\mathbf{m}}_0[\texttt{Node}] = \langle zv_1 + zv_2 + zv_3\rangle$$

where $|zv_1| = |zv_3| = 1$, $|zv_2| = 2$. This symbolic marking represents a number of equivalent SN markings (six if N = 4) including $\mathbf{m}_{G_0}$ defined at the end of the last section. Observe that colors v_2, v_3 are now folded into zv_2.

The SRG is *directly* built from $\widehat{\mathbf{m}}_0$ through a symbolic firing mechanism. Leaving out technical details, a *symbolic instance* of transition t_{r_1} (Figure 2) is: $(n_1 = zv_3,\ n_2 = zv_1,\ n_3 = zv_2)$. This symbolic instance, which folds two color instances, is enabled in $\widehat{\mathbf{m}}_0$; when firing, it leads to the symbolic marking [5]:

$$\langle zv_3, zv_1, \mathrm{L}_1\rangle + \langle zv_1, zv_2, \mathrm{L}_1\rangle + \langle zv_3, zv_2, \mathrm{L}_1\rangle,\ |zv_2| = 2$$

The SRG of the simple example we are considering contains 10 nodes, one of which absorbing, against the 16 nodes of the corresponding RG.

When considering large graphs, the reduction achieved with the SRG in terms of states/edges may be dramatic. In particular, if a GTR adds new nodes to the encoded graph, GTR's symbolic firing instances may fold together a huge number of ordinary ones. The reason is that added nodes correspond to class-N colors bound to *fresh* transition variables, like n_4 in Figure 1 (R6).

A *canonical representative* for SM permits a syntactical comparison between SM. In the context of graph encoding, bringing an SM into its canonical form has

[5] we only refer to place Edge, because the marking of `Node` doesn't change

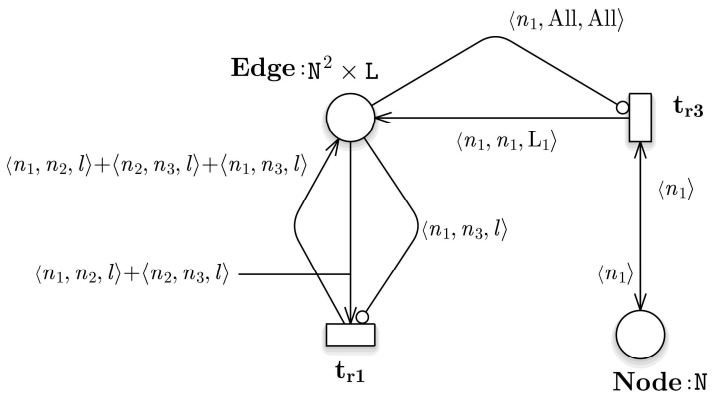


Figure 2: GTS composed of rules R1,R3 of Figure 1

more or less the same complexity as checking graph isomorphism. We believe that the legacy `GreatSPN` algorithm computing SM canonical representatives (which in addition, for performance analysis purposes, calculates the size of SM by enumerating color permutations) might be significantly improved by using structural analysis outcomes. This is a part of ongoing work.

In the rest of the paper, to keep notations simple, we shall refer to ordinary markings/transition instances.

5. Exploiting SN Structural Analysis: Application Examples

In this section, we aim at showing the potential of SN structural analysis in graph transformation field, by means of a simple example of GTS. We introduce/use (symbolic) structural relations and (colored and numerical) semiflows.

Symbolic structural analysis is a promising, effective approach to formal verification of properties, with a lot of positive effects also on traditional methods based on state-space exploration and model-checking.

In Section 3.3, we have set conditions on arc functions characterizing SN Graph Transformation Rules. These conditions, which involve base functional operators on bag-expressions, can be automatically and efficiently checked with the `SNexpression` tool.

The SN structural calculus allows you not only to validate GTRs, but also to check general properties of a GTS, e.g., figure out which rules are in conflict (as a consequence, may concurrently apply), mutually exclusive, causally connected, and so forth. True concurrent or partial-order approaches have been widely treated in graph rewriting and PN literature: we are not interested in a theoretical discussion about these specific topics, rather we aim at showing the potential of SN structural analysis in these and related research areas.

5.1. Symbolic Structural Relations

A (symbolic) structural relation between SN nodes e, e', defined as $\mathcal{R}(e, e') : \mathcal{D}(e') \to 2^{\mathcal{D}(e)}$, maps any instance b' of e' to the set of instances of e related with b'. Formally, structural relations are expressions involving arc functions and a base set of functional operators: transpose, sum, intersection, difference, support, and composition.

Two base relations, $\mathrm{Rb}[t, p]$, $\mathrm{Ab}[t, p]$, and others higher-level, are listed in Table 2 with their formulae. $\mathrm{Rb}[t, p]$ (*Removed by*), given an element c of the color domain of place p, results in the set of instances of t that withdraw c from p; $\mathrm{Ab}[t, p]$ (*Added by*) is dual, given a color c results in the set of instances of t that put c in p. An intuitive explanation of the others follows.
(Asymmetric) Structural Conflict: two transition instances (t, b) and (t', b') are in a conflict relation in marking $\mathbf{m}$ if the firing of the former disables the latter. The structural conflict relation (SC) sets the necessary conditions for a conflict: $SC(t, t')$ maps an instance b' of t' to the set of (all and only) the colour instances b of t that may disable (t', b'). A conflict arises because (t, b) withdraws tokens from an input place of t' or it adds tokens to an inhibitor place of t'. The SC formula is the summation over all local conflicts possibly caused by the situations described above. Different instances of the same transition may be conflicting: the corresponding, one-argument SC formula, therefore, uses the identity function as a final subtrahend.
Structural Causal Connection: two transition instances (t, b) and (t', b') are in causal connection in a marking $\mathbf{m}$ if the firing of the former causes the enabling of the latter. The structural causal connection (SCC) relation sets the necessary conditions for causal connection: $SCC(t, t')$ maps an instance b' of t' to the set of the instances b of t that may cause the enabling of (t', b'). This situation occurs if (t, b) adds tokens to an input place of t' or it withdraws tokens from an inhibitor place of t'.
Structural Mutual Exclusion: two instances (t, b) and (t', b') are in structural mutual exclusion (SME) if whenever one is enabled the other is not, and vice-versa. This case occurs when there is a place p which is both an input place for t and an inhibitor place for t', and the number of tokens of a certain color required for the enabling of t is at least equal to the upper-bound enforced by the inhibitor arc-function for tokens of that color. The symmetric relation $SME(t, t')$ maps an instance b' of t' to the set of instances of t that cannot be enabled when (t', b') is. The SME formula given in Table 2 works if all involved input/inhibitor arc-functions are type-set, as in most of SN models presented in this paper. Note that the formula is slightly different if $t = t'$. [25] treats the general case of SME.

Applications. Assume that we want to determine which rules (described by SN) of a GTS may simultaneously apply. By using the SN structural calculus, we can give a precise answer. First of all, we should partition the whole set of GTS rules' instances into (symbolic) conflict sets, each representing all the instances which may conflict, directly or indirectly. And even though, in general, this

Table 2: Symbolic structural relations in SN

$\mathrm{Ab}[t,p]$	$=$	$\overline{\mathrm{W}^+[p,t]}^t$	
$\mathrm{Rb}[t,p]$	$=$	$\overline{\mathrm{W}^-[p,t]}^t$	
$SC(t,t')$	$=$	$\sum_p \mathrm{Rb}[t,p] \circ \overline{\mathrm{I}[t',p]} + \mathrm{Ab}[t,p] \circ \overline{\mathrm{H}[t',p]}$	$t \neq t'$
$SC(t)$	$=$	$\left(\sum_p \mathrm{Rb}[t,p] \circ \overline{\mathrm{I}[t,p]} + \mathrm{Ab}[t,p] \circ \overline{\mathrm{H}[t,p]}\right) - \mathit{Ide}_{\mathcal{D}(t)}$	
$SCC(t,t')$	$=$	$\sum_p \mathrm{Ab}[t,p] \circ \overline{\mathrm{I}[t',p]} + \mathrm{Rb}[t,p] \circ \overline{\mathrm{H}[t',p]}$	
$SME(t,t')$	$=$	$\sum_p \overline{\mathrm{I}[t,p]}^t \circ \overline{\mathrm{H}[t',p]} + \overline{\mathrm{H}[t,p]}^t \circ \overline{\mathrm{I}[t',p]}$	$t \neq t'$
$SME(t)$	$=$	$(\sum_p \overline{\mathrm{I}[t,p]}^t \circ \overline{\mathrm{H}[t,p]}) - \mathit{Ide}_{\mathcal{D}(t)}$	

requires calculating the symmetric and transitive closure of SC relation [25], in the following examples we can use a smarter technique.

To illustrate the concept, let us consider the GTS in Figure 2. The two rules are potentially in conflict due to place `Edge`, which is (reciprocally) an output place for one and an inhibitor place for the other. There are no potential conflicts due to shared input places since the corresponding expressions of $\mathrm{Rb}[t,p]$ turn out to be null (by the way, a composition involving null operand results in turn null). As for *added by*, we get the following non-null expressions: (in the rest of the section, function supports are implicitly used): $\mathcal{D}(t_{r_1}) = \mathrm{N}^3 \times \mathrm{L}$, $\mathcal{D}(t_{r_3}) = \mathrm{N}$

$$\mathrm{Ab}[t_{r_1}, \texttt{Edge}] = \langle n_1, \mathit{All}, n_2, l\rangle \qquad \mathrm{Ab}[t_{r_3}, \texttt{Edge}] = \langle n_1\rangle[n_1 = n_2 \wedge l \in \mathrm{L}_1]$$

The expression on the left says that a color tuple $\langle c_1, c_2, lb_i\rangle$ is put in place `Edge` by *any* instance $\langle c_1, *, c_2, lb_i\rangle$ of t_{r_1}. The other expression instead says that a color tuple $\langle c_1, c_1, lb_1\rangle$ is put in place `Edge` by *the* instance $\langle c_1\rangle$ of t_{r_3}.

According to the formula in Table 2 and the tuple-transpose algorithm [24], we obtain, after some rewriting:

$$\begin{aligned} SC(t_{r_1}, t_{r_3}) &= \langle n_1, \mathit{All}, n_2, l\rangle \circ \langle n_1, \mathit{All}, \mathit{All}\rangle && \equiv \langle n_1, \mathit{All}, \mathit{All}, \mathit{All}\rangle \\ SC(t_{r_3}, t_{r_1}) &= \langle n_1\rangle[n_1 = n_2 \wedge l \in L_1] \circ \langle n_1, n_3, l\rangle && \equiv \langle n_1\rangle[n_1 = n_3 \wedge l \in \mathrm{L}_1] \end{aligned}$$

Once again, we can supply an intuitive explanation: $SC(t_{r_1}, t_{r_3})$ indicates that an instance $\langle c_1\rangle$ of t_{r_3} may be in conflict with any instance $\langle c_1, *, *, *\rangle$ of t_{r_1}, $SC(t_{r_3}, t_{r_1})$ instead indicates that an instance $\langle c_1, *, c_1, lb_1\rangle$ of t_{r_1} is in conflict with the instance $\langle c_1\rangle$ of t_{r_3}.

The SC relation points out potential conflicts. We can enhance the previous outcome by computing SME: `Edge` is both an input and an inhibitor place for t_{r_1}, and inhibitor for t_{r_3}. According to the formula in Table 2, we obtain:

$$\begin{aligned} SME(t_{r_1}, t_{r_3}) =& \big(\langle n_1, n_2, \mathit{All}, l\rangle + \langle \mathit{All}, n_1, n_2, l\rangle\big) \circ \langle n_1, \mathit{All}, \mathit{All}\rangle \\ \equiv& \langle n_1, \mathit{All} - n_1, \mathit{All}, \mathit{All}\rangle + \langle \mathit{All}, n_1, \mathit{All}, \mathit{All}\rangle \end{aligned}$$

$SME(t_{r_3}, t_{r_1})$ turns out to be $\langle n_1\rangle + \langle n_2\rangle$, i.e., $SME(t_{r_3}, t_{r_1}) \equiv SME(t_{r_1}, t_{r_3})^t$, because SME is symmetric. But what matters is that:

$$SC(t_{r_1}, t_{r_3}) \subseteq SME(t_{r_1}, t_{r_3}) \ \wedge \ SC(t_{r_3}, t_{r_1}) \subseteq SME(t_{r_3}, t_{r_1})$$

That is, any conflicting instances of t_{r_1} and t_{r_3} are mutually exclusive. In other words, these two rules are potentially concurrent.

We can also extend this result to instances of the *same* GTR. According to the formula in Table 2, we get ($Ide_{\mathcal{D}(t_{r_1})} = \langle n_1, n_2, n_3, l\rangle$):

$$\begin{aligned} SC(t_{r_1}) = &\langle n_1, All - n_1, n_3, l\rangle[n_1 = n_2 \wedge n_1 \neq n_3] + \langle n_2, All, n_3, l\rangle[n_1 \neq n_2] + \\ &\langle n_1, All, n_1, l\rangle[n_1 = n_2 \wedge n_1 \neq n_3] + \langle n_1, n_2, n_2, l\rangle[n_1 \neq n_2 \wedge n_2 \neq n_3] + \\ &\langle n_1, All - n_1, n_1, l\rangle[n_1 = n_2 \wedge n_1 = n_3] + \langle n_1, All - n_2, n_2, l\rangle[n_1 \neq n_2] \end{aligned}$$

The expression for mutually exclusive instances of GTR t_{r_1} is:

$$\begin{aligned} SME(t_{r_1}) = &\langle n_1, All - n_1, n_3, l\rangle[n_1 = n_2 \wedge n_1 \neq n_3] + \langle n_2, All, n_3, l\rangle[n_1 \neq n_2] + \\ &\langle n_1, All, n_1, l\rangle[n_1 = n_2 \wedge n_1 \neq n_3] + \langle n_1, n_2, n_2, l\rangle[n_1 \neq n_2 \wedge n_2 \neq n_3] + \\ &\langle n_1, All - n_1, n_1, l\rangle[n_1 = n_2 \wedge n_1 = n_3] + \langle n_1, All - n_2, n_2, l\rangle[n_1 \neq n_2] \end{aligned}$$

Also in this case, it holds $SC(t_{r_1}) \subseteq SME(t_{r_1})$, i.e., different instances of t_{r_1} are potentially concurrent. As for t_{r_3}, there are no auto-conflicts.

We performed all the calculations with `SNexpression` and, on average, it took around a few dozen ms per test.

Detecting GTS rule instances which may run independently from each other would generally require more sophisticated calculations. Related concerns, however, are out of the scope of this paper.

5.2. Checking Semiflows

An opportunity offered by SN is the automated verification of structural invariants involving place marking (P-invariants, or semi-flows) or transition sequences (T-invariants). These invariants do not take account of inhibitor arcs and priorities. However, they allow you to formally and efficiently check interesting properties, orthogonal or complementary to structural relations. Focusing on P-invariants, we can distinguish between colored (symbolic) and numerical ones. The former give us both qualitative and quantitative information, hence are of particular interest.

The structural calculus implemented in `SNExpression` tool can be used to verify whether a P-indexed vector $\mathbf{i}$ of bag-expressions $\mathbf{i}[p] : \mathcal{D}(p) \rightarrow Bag[\mathcal{D}_{inv}]$ is a *coloured* P-invariant. $\mathcal{D}_{inv}$ is the invariant's color domain.

Vector $\mathbf{i}$ is a P-invariant (or semi-flow) if and only if, for each transition t, $\sum_{p \in P} \mathbf{i}[p] \circ (1 \cdot \mathrm{O}[p,t] - 1 \cdot \mathrm{I}[p,t]) \equiv \epsilon$ [6]. The invariant expression we obtain from the SN initial marking is: $\sum_p \mathbf{i}[p](\mathbf{m}_0[p])$.

Consider the GTS composed of two simple rules shown in Figure 3: one (t_{r_1}) replaces everywhere edge label lb_1 with lb_2, the other (t_{r_2}) deletes edge-loops whose label is neither lb_1 nor lb_2. In this fairly common case, the only non-null invariant entry refers to place `Edge`, which encodes the connected components

[6] negative terms in a linear combination impact on the final outcome

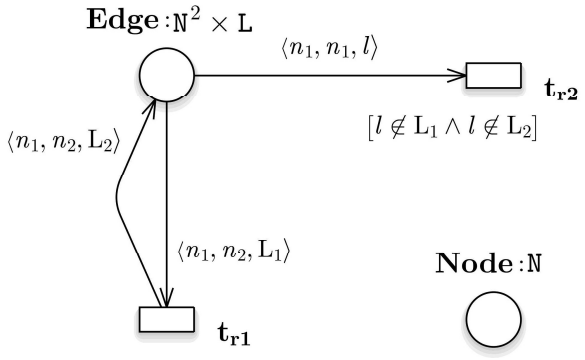


Figure 3: GTS composed of two rules

of a graph. Let X denote the neutral color, $x : \mathrm{X}$, $l : \mathrm{L}$, $n_1, n_2 : \mathrm{N}$. Here are some possible invariants:

$$\begin{aligned} pinv_1 &:= \mathrm{N}^2 \times \mathrm{L} \to \mathrm{X} && \langle x\rangle[l \in \mathrm{L}_1 \vee l \in \mathrm{L}_2] \\ pinv_2 &:= \mathrm{N}^2 \times \mathrm{L} \to \mathrm{N}^2 && \langle n_1, n_2\rangle[n_1 \neq n_2] \\ pinv_3 &:= \mathrm{N}^2 \times \mathrm{L} \to \mathrm{N}^2 \times \mathrm{L} && \langle n_1, n_2\rangle[l \in \mathrm{L}_1 \vee l \in \mathrm{L}_2] \end{aligned}$$

The three expressions above satisfy the invariance property based on composition, therefore, are P-invariants. You can check it with `SNExpression` (which adopts a syntax very close to that used here) in just a few ms. Notice that these invariants depend on the size of subclasses L_1, L_2: the composition involving $pinv_1$ and t_{r_1}, e.g., results in $|\mathrm{L}_2| \cdot \langle x\rangle - 1 \cdot |\mathrm{L}_1| \cdot \langle x\rangle$, which is equal to ϵ if, as we are assuming, the two subclasses have the same size.

These invariants have a simple interpretation: $pinv_1$ means that the *total* of edges with label lb_1 or lb_2 is preserved; $pinv_2$ means that edges whose source differs from the target are preserved, without considering their labels; $pinv_3$ is like $pinv_1$ but more precise: it tells that edges with label lb_1 or lb_2, no matter which, are preserved.

5.2.1. Numerical Semiflows

Due to a property of SN arc functions, we may also check for numerical semiflows (or even minimal semiflow bases), typical of classical low-level PN. The *size* of a bag is the sum of multiplicities of the bag's elements.

Property 3 A SN arc-function $F : \mathcal{D}(t) \to Bag[\mathcal{D}(p)]$ may be rewritten as $\sum_i \lambda_i \cdot T_i[g_i]$, $\lambda_i \in \mathbb{N}$, where tuple guards are mutually exclusive and:

$$\forall i\ \exists k \in \mathbb{N}\ \forall b \in \mathcal{D}(t) :\ g_i(b) = true \Rightarrow |T_i(b)| = k$$

In other words, any arc-function can be rewritten as a weighted sum of function-tuples that map to constant-size bags, but for colour instances making the tuple guards false. Consider $\langle \mathrm{C}_i - c \rangle$, which, depending on whether the colour bound to c belongs to subclass C_i or not, has size $|\mathrm{C}_i| - 1$ or $|\mathrm{C}_i|$: we may rewrite $\langle \mathrm{C}_i - c \rangle \rightarrow \langle \mathrm{C}_i - c \rangle [c \in \mathrm{C}_i] + \langle \mathrm{C}_i \rangle [c \notin \mathrm{C}_i]$, according to Property 3. This is the normal form used by `SNexpression` for bag-expressions.

As a consequence, by embedding function-tuple guards in transition guards (always possible), we can split any SN transition into an equivalent set of mutually-exclusive replica[7], whose arc-functions are "constant-size". In analogy with low-level PN, a $|P| \cdot |T|$ matrix $\mathbf{H}$ of $\mathbb{Z}$ values is thus defined, whose $[p,t]$ entry is $|\mathrm{O}[p,t](b)| - |\mathrm{I}[p,t](b)|$, for any $b \in \mathcal{D}(t)$. Any non-null P-vector $\mathbf{y}$ which is a positive integer solution of the matrix product $\mathbf{y} * \mathbf{H} = \mathbf{0}$ is a semiflow, expressing a conservative law for tokens flowing through the places corresponding to semiflows's non-zero entries, which abstracts from tokens' color. We say a place p is covered by a semiflow is the semiflow's p-entry is non-null. We say an SN is covered if every place is. One such an SN is *color-safe*.

As for the GTS example of this section, place `Edge` is not covered by numerical semi-flows. We shall exploit (numerical) semiflows in the next section.

6. SN Composite Graph Transformation Rules

In this section, we define a more structured type of GTR able to emulate complex graph transformations (some of which) peculiar to algebraic, rule-based models. We use SN *subnets* with a well-defined layout.

Until now, we have considered context-dependent graph transformations. Some typical operations on graphs, however, apply in undetermined contexts: an example treated here is node deletion with simultaneous removal of incident edges, natively supported by the single-pushout [29] approach, but not by the DPO. Other examples, considered later, are node merge/split.

It is not possible to represent these operations on graphs through a single SN transition, due to the low data abstraction provided by SN arc-function syntax. Using the *All* function is not the solution, since it yields all the elements of a given color class.

The workaround consists of representing a GTR through an SN subnet, called *composite rule*, composed of one observable transition, which operates context-dependent changes and triggers a finite sequence of immediate transition instances in charge of performing changes in unspecified contexts.

Composite rules meet a quite simple structural pattern making it possible their efficient validation. Some subnet places have a neutral domain, their incident weight-one arcs do not carry any annotation, for simplicity (it should be $\langle All \rangle$). We denote with ${}^\bullet t$, $t^\bullet$, the set of places linked to a transition t through non-null input, output arc-functions, respectively.

[7] that is, t may be equivalently "split" into a set $EQ = \{t_i\}$, such that $\mathcal{D}(t) = \mathcal{D}(t_i)$, $\forall t_i \in EQ$, and $\mathbf{m}[t,b\rangle\mathbf{m'} \Leftrightarrow \exists t_i \in EQ : \mathbf{m}[t_i,b\rangle\mathbf{m'}$, $\forall \mathbf{m}, b \in \mathcal{D}(t)$

Definition 4 (SN Composite GTR) A composite rule is a SN including, among the others, places `Edge`, `Node` (whose color domains have been previously defined), neutral places `Start`, `Trigger`, one observable transition t_r, and a set lmm of immediate transitions including *end*, such that:

- $\texttt{Start} \in end^\bullet \cap {}^\bullet t_r$; $\texttt{Trigger} \in t_r{}^\bullet$
- $\forall t \in \mathsf{lmm} : \texttt{Trigger} \in {}^\bullet t \qquad \forall t \in \mathsf{lmm} - \{end\} : \texttt{Trigger} \in t^\bullet$
- $\forall t \in \mathsf{lmm} - \{end\} : t \; SME \; end$
- the SN is covered by (numerical) semiflows
- every SN transition satisfies Lemma 1

Places `Start`, `Trigger` are covered by a semi-flow, by construction. The coverage requirement for the other places is to ensure color-safeness. The mutual exclusion between *end* and any other immediate transition guarantees that the rule's behaviour is well-defined. Whenever *end* fires, it disables all the other immediate transitions, bringing back the SN to its initial state.

As an example, Figure 4 shows an SN composite rule deleting a graph node without successors but itself, together with all incident edges. Transition t_r checks for the application condition by matching a node v_i, then transition *delInEdge* removes all edges with target v_i, finally *end* deletes v_i by respecting the dangling-edge condition. The only extra place `NoSucc`, initially marked with any class N colour, is clearly covered by a semiflow. The mutual exclusion between *end* and *delInEdge* is ensured by the arc-function H[`Edge`, *end*].

We need to slightly change the notion of GTS to include composite rules.

Definition 5 (Generalized SN Graph Transformation System) Let G_0 be a graph, $\mathcal{R}_e$ be a set of (SN) Graph Transformation Rules (Definition 2) and $\mathcal{R}_c$ a set of disjoint SN implementing composite rules (Definition 4). The associated GTS is the SN model obtained by sharing places `Node` and `Edge` among $\mathcal{R}_e$ and $\mathcal{R}_c$, with $\mathbf{m}_0[\texttt{Node}], \mathbf{m}_0[\texttt{Edge}]$ encoding G_0, and $\{\texttt{Start}_i\}$ places of $\mathcal{R}_c$ initially holding one neutral token.

The SN Reachability Graph (precisely, its projection on places `Node` and `Edge`) defines the GTS state-transition system.

7. A Comparison with the DPO Approach

In the second part of the paper, we compare the SN-based graph transformation approach with a classical, rule-based one. We consider basic graph rewriting, in particular, the algebraic approach based on double-pushout (DPO), one of the most well-known (Turing-complete) approaches. We do not use the original, categorical pushout concepts, but we follow a somewhat operational

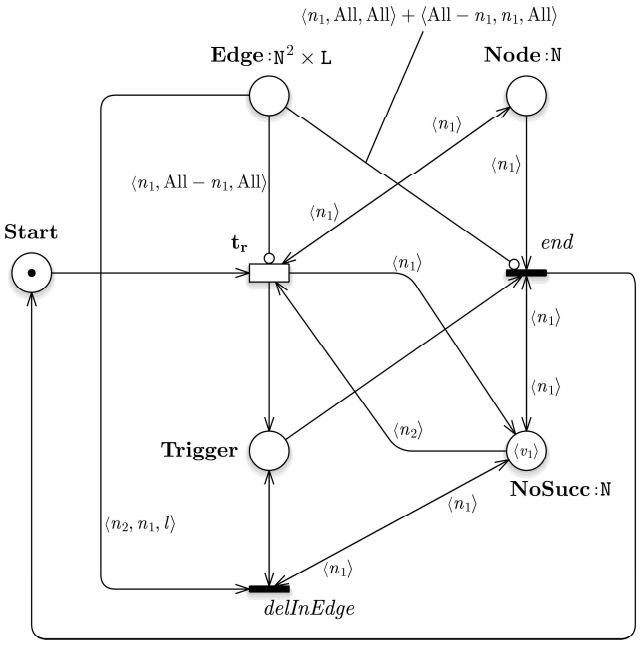


Figure 4: Composite rule: removal of a node without proper successors

line. We use well-known definitions which solely refer to the concepts of graph (total) morphism and gluing, formally presented in the background.

The aim is twofold: to provide the SN-based graph transformation approach with a theoretically sound semantic characterization, and to promote interoperability among different modelling formalisms and tools.

We follow a constructive approach: given a DPO rule, we show how to formally derive (step-by-step) the color annotations of the corresponding SN GTR, first considering "injective" DPO rules, then non-injective ones.

After recalling the base concepts of DPO rules (Section 7.1), we formalize the translation of injective rules in the (widespread) case of an injective match of rules (Section 7.2). Then we set a precise relationship between rule matches and SN transition instances, also considering non-injective matches (Section 7.3). Finally, we deal with non-injective DPO rules (Section 7.4) and shortly discuss on the reverse direction SN $\rightarrow$ DPO (Section 7.5).

7.1. Basic Definitions

DPO Graph Transformation Rule. A DPO rule r is composed of three graphs, L, I, R (left-hand side, interface, and right-hand side, respectively) and two morphisms: $L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$.

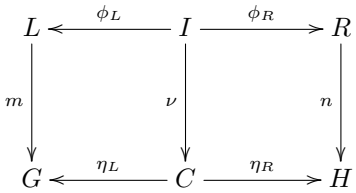
The elements of graph L not belonging to the image of ϕ_L are said *obsolete*, whereas the elements of R not in the image of ϕ_R are said *fresh*. If both

morphisms are injective we say that the rule is *injective*.

The application of an injective rule has an intuitive explanation: once a match of the rule's left-hand side is found in a host graph G, the rule applies by removing obsolete nodes/edges, adding fresh elements, and preserving I, which plays the role of a solid attachment point.

However, if ϕ_L or ϕ_R is not injective, the previous interpretation has to be corrected because of some merge/split effect (depending on whether ϕ_R or ϕ_L is non-injective, respectively). The general semantics of graph transformation makes use of the graph gluing concept.

Graph Transformation. Given $r = (L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R)$, a graph G is transformed by r into a graph H (written $G \overset{r}{\Rightarrow} H$) if there is a graph C (context) and a morphism $\nu : I \to C$, such that $G \cong L +_{\phi_L,\nu} C$ and $H \cong R +_{\phi_R,\nu} C$.



We illustrate this on the commutative diagram above, where the morphisms m and n are called match and co-match, respectively. In other words, we search for an unknown graph C (called *context*) such that the host graph G is the gluing of L and C over I. If such a context does exist, we rewrite G to the graph obtained by gluing C and R over I.

Many papers make strong assumptions on rules. Most of them deal with injective rules and extend this assumption to rule match. Almost all assume that the morphism ϕ_L is injective. We shall first consider the base case of an injective rule/match then we shall treat non-injective m and ϕ_R.

Differently from term rewriting, the existence of a match of the left-hand side of a rule in a host graph doesn't ensure that the rule can apply. Two more conditions guarantee it, the dangling-edge condition (if we remove a node, then we contextually remove all incident edges) and the identification condition (a match may only identify elements which are preserved). We call the sum of these two conditions gluing condition.

Property 4 Let $r = (L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R)$ be a rule and $m : L \to G$ be a graph morphism, such that m_E is injective. Then a context C and a morphism $\nu : I \to C$ such that $G \cong L +_{\phi_L,\nu} C$ do exist if and only if

- $\forall v \in N_L$ such that $m(v)$ is incident to $e \in E_G - m_E(E_L)$: $v \in \phi_L(N_I)$
- $\forall v_1, v_2 \in N_L$, $v_1 \neq v_2 : m(v_1) = m(v_2) \Rightarrow v_1, v_2 \in \phi_L(N_I)$

Informally, the dangling condition (the 1st one) says that every node of L whose image is incident to an edge of G which is not in the image of m is not obsolete.

If ϕ_L is injective and m satisfies the gluing condition, the context C and the morphism ν are unique. Rule r thus applies deterministically (up to graph isomorphism) and we use the notation $G \overset{r,m}{\Longrightarrow} H$.

Figure 5 gives an example of injective DPO rule and corresponding graph transformation. We shall refer to it to illustrate some basic concepts/notations.

By convention, we describe a morphism $f : G_1 \to G_2$ by associating identifiers 1 through $|N_{G_1}|$ with nodes of the source graph (we may thus speak of i^{th} node), and pair-wise disjoint, non-empty sets of values in the same range to nodes belonging to the image of G_1. In the case of injective morphism, all these sets are singletons.

7.2. Mapping Injective DPO Rules to SN Graph Transformation Rules.

Let t_r denote the transition corresponding to $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$. The following steps formalize the functions on the arcs connecting t_r to places `Node`, `Edge` and the guard of t_r. In the sequel, $\bigwedge_{\emptyset}(\ldots) \equiv true$, whereas $\sum_{\emptyset}(\ldots) \equiv \epsilon$.

Procedure 1 (SN translation of an injective DPO rule - injective match)

1. The set $Var(t_r) = \{n_i, i : 1 \ldots \mathrm{k}\},\ k \in \mathbb{N}^+$, of type-N variables (projections) used in arc functions is *partitioned* into:

$$Var(t_r) = Var_I \cup Var_{obs} \cup Var_{fresh}$$

   $Var_I = \{n_i, i : 1 \ldots |N_I|\}$ corresponds to the set of interface's nodes, whereas Var_{obs} and Var_{fresh} are *isomorphic* to $N_L - \phi_L(N_I)$ and $N_R - \phi_R(N_I)$, respectively (the *obsolete* and *fresh* nodes). By convention, if $n_i \in Var_{obs}$ and $n_j \in Var_{fresh}$, then $i < j$. Let $var : N_I \cup N_L \cup N_r \to Var(t_r)$ map each graph node to the corresponding variable: $var(N_I) = var(\phi_L(N_I)) = var(\phi_R(N_I))$.
2. The arc functions involving place `Node` are

$$\mathrm{I}[\texttt{Node}, t_r] = \sum_{n_i \in Var_I \cup Var_{obs}} \langle n_i \rangle \qquad \mathrm{O}[\texttt{Node}, t_r] = \sum_{n_i \in Var_I \cup Var_{fresh}} \langle n_i \rangle$$

$$\mathrm{H}[\texttt{Node}, t_r] = \sum_{n_i \in Var_{fresh}} \langle n_i \rangle$$

3. Having mapped graph nodes to transition's variables, each $e \in E_I \cup E_L \cup E_R$, with $l(e) = lb_k$, is consequently identified by a tuple $\langle n_i, n_j, \mathrm{L}_k \rangle$, where $n_i = var(s(e))$, $n_j = var(t(e))$. Since we are considering multi-graphs, we can represent the sets E_I, E_L, E_R as (symbolic) bags[8]: $bag_{E_I} \in Bag[{Var_I}^2 \times \mathrm{L}]$, $bag_{E_L} \in Bag[(Var_I \cup Var_{obs})^2 \times \mathrm{L}]$, $bag_{E_R} \in Bag[(Var_I \cup Var_{fresh})^2 \times \mathrm{L}]$. Since ϕ_R and ϕ_L are injective morphisms, $bag_{E_I} \leq bag_{E_L} \wedge bag_{E_I} \leq bag_{E_R}$.

[8] They can be seen both as formal bags of functions and bag-functions

4. The arc functions involving place `Edge` are:

$$\mathrm{I}[\texttt{Edge}, t_r] = bag_{E_L} \qquad \mathrm{O}[\texttt{Edge}, t_r] = bag_{E_R}$$
$$\mathrm{H}[\texttt{Edge}, t_r] = H_1 \quad \text{(see Lemma 1, C 3.)}$$

5. The guard $g(t_r)$ is a *conjunctive* form which contains the following predicates:

$$\forall n_i, n_j \in Var_{fresh},\ i \neq j :\ n_i \neq n_j$$
$$\forall n_i \in Var_{obs}\ \forall n_j \in Var_{obs} \cup Var_I,\ i \neq j :\ n_i \neq n_j$$

We have mechanically obtained the SN in Figure 6 (a) from the DPO rule in Figure 5, by applying Procedure 1. In this case, we have: $Var_I = \{n_1, n_2\}$, $Var_{obs} = \emptyset$, $Var_{fresh} = \{n_3\}$.

Property 5 A transition t_r defined according to Procedure 1 is a SN Graph Transformation Rule (Definition 2).

PROOF. The 1st condition of Lemma 1 holds since $(\sum_{n_i \in Var_{fresh}} \langle n_i \rangle)[g(t_r)] \leq \langle All \rangle$ and $\mathrm{W}^+[\texttt{Node}, t_r] = (\sum_{n_i \in Var_{fresh}} \langle n_i \rangle)[g(t_r)] - \ldots$. As for the 2nd condition, it is sufficient to observe that $\langle n_1 + n_2 \rangle \circ \mathrm{O}[\texttt{Edge}, t_r] \subseteq Var_I \cup Var_{fresh} = \overline{\mathrm{O}[\texttt{Node}, t_r]}$. The 3rd and last condition is enforced by step 4. ■

Enabled instances of transition t_r meet the property below.[9].

Property 6 Let t_r be the SN transition obtained by translating an injective DPO rule r according to Procedure 1 and $\mathbf{m}$ a graph-encoding.

If (t_r, b) is enabled in $\mathbf{m}$ then $\forall n_i, n_j \in Var(t_r),\ i \neq j :\ n_i(b) \neq n_j(b)$.

PROOF. Since $\mathbf{m}[\texttt{Node}]$ is type-set, due to $\mathrm{I}[\texttt{Node}, t_r], \mathrm{H}[\texttt{Node}, t_r]$ (Step 3), enabled instances of t_r verify, other than $g(t_r)$ (Step 5), these *implicit* predicates:

$$\forall n_i, n_j \in Var_I,\ i \neq j :\ n_i(b) \neq n_j(b)$$
$$\forall n_i \in Var_{fresh}\ \forall n_j \in Var_I \cup Var_{obs} : n_i(b) \neq n_j(b).$$

■

Sometimes, we may rewrite the color annotations of the SN transition resulting from Procedure 1 in a somewhat simpler form.

Hereinafter, we shall use two extra symbols: $Var_I^* \subseteq Var_I$ includes all and only Var_I variables occurring on $\mathrm{I}[\texttt{Edge}, t_r]$ whereas $Var_I^- = Var_I - Var_I^*$.

[9] Given a transition instance b, we may equivalently write $b :\ (n_i = v_i, \ldots)$, or $n_i(b) = v_i$, depending on whether we see n_i as a variable or a projection

Property 7 Let t_r be the SN translation of an injective rule $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$, according to Procedure 1. If we erase Var_I^* from $\mathrm{I}[\mathtt{Node}, t_r]$, $\mathrm{O}[\mathtt{Node}, t_r]$, and introduce in $g(t_r)$ the predicates $n_i \neq n_j$, $\forall n_i \in Var_I^*, n_j \in Var_I$, $i \neq j$, we get an equivalent GTR.

Proof. Let t'_r be obtained by rewriting t_r, as indicated. By construction, t'_r has the same color domain as t_r, and $\mathrm{W}^+[p, t_r] = \mathrm{W}^+[p, t'_r]$, i.e., any two instances (t_r, b) and (t'_r, b), when firing, have the same effect. If (t_r, b) is enabled, then also (t'_r, b) is, based on the aforementioned implicit predicates (Property 6). On the other side, if (t'_r, b) is enabled in $\mathbf{m}$ (therefore, $\mathrm{I}[\mathtt{Edge}, t'_r](b) \leq \mathbf{m}[\mathtt{Edge}]$), we know that $F(b) = \sum_{n_i \in Var_I^*} n_i(b) \leq \langle n_1 + n_2 \rangle \circ \mathbf{m}[\mathtt{Edge}] \leq \mathbf{m}[\mathtt{Node}]$ (Definition 1, graph-encoding), with $F(b)$ being type-set due to the guard. Given that $\mathrm{I}[\mathtt{Node}, t_r](b)$ is equal to the disjoint sum $\mathrm{I}[\mathtt{Node}, t'_r](b) + F(b)$ (both terms are less than or equal to $\mathbf{m}[\mathtt{Node}]$), (t_r, b) is enabled in $\mathbf{m}$. ■

Figure 6 (b) shows the equivalent (simpler) translation of DPO rule in Figure 5, according to Property 7. In this case, $Var_I^* = \{n_1, n_2\}$.

An example of mechanical translation of an injective DPO rule involving node removal, according to Procedure 1, is shown in Figures 7,8. In that case, we have: $Var_I = Var_I^* = \{n1\}$, $Var_{obs} = n_2$, $Var_{fresh} = \emptyset$. You can see that the inhibitor arc function ensuring the dangling-edge condition takes account of the (only) edge incident to the removed node, which is contextually withdrawn by the rule. This corresponds to the weight-two term in the arc-function expression.

We may carry out a comparison between an injective r: $L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ and the corresponding SN transition t_r through the diagram below, which obscures, however, the role played by inhibitor arcs and transition guard:

$$\binom{\mathrm{I}[\mathtt{Node}, t_r]}{bag_{E_L}} \geq \binom{\mathrm{I}[\mathtt{Node}, t_r] \cap \mathrm{O}[\mathtt{Node}, t_r]}{bag_{E_I}} \leq \binom{\mathrm{O}[\mathtt{Node}, t_r]}{bag_{E_R}}$$

7.3. Rule Match and Graph Transformation

In order to define the semantics of a transition t_r representing a DPO rule r, we need to formalize a graph morphism $m : L \to G$ (hereinafter, simply *match*) in terms of t_r. We first consider injective matches, to which we have so far implicitly referred. Then, we treat the more general, even if less frequent, case.

We set an intuitive relationship between $m : L \to G$ and an *equivalence class* of instances of t_r, holding independently on whether m is injective or not. Let $b_1, b_2 \in \mathcal{D}(t_r) = \mathrm{N}^{|Var_I|+|Var_{obs}|+|Var_{fresh}|}$: we write $b_1 \equiv b_2$ if and only if $\forall n_i \in Var_I \cup Var_{obs} : \ n_i(b_1) = n_i(b_2)$.

In a graph encoding context, equivalent instances behave in the same way.

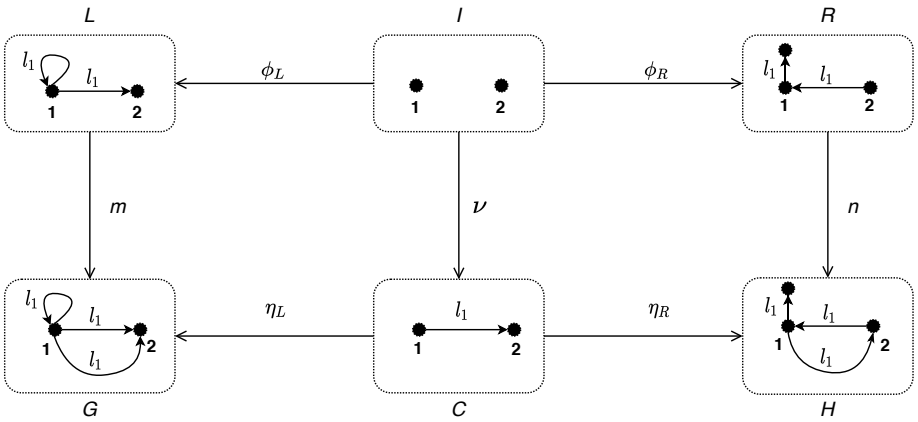
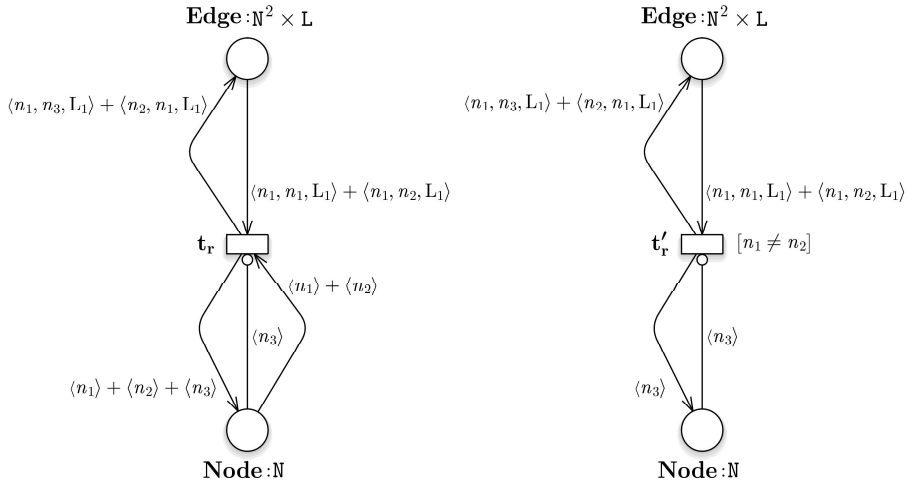


Figure 5: Injective DPO rule and graph transformation



(a) Original

(b) Equivalent

Figure 6: SN Translation(s) of DPO rule in Figure 5

Property 8 Let t_r be the SN transition translating an injective DPO rule r, $b_1, b_2 \in \mathcal{D}(t_r)$, with $b_1 \equiv b_2$, and $\mathbf{m}_G$ a graph-encoding.

If $\mathbf{m}_G[t_r, b_1\rangle \mathbf{m}_H$ then $\mathbf{m}_G[t_r, b_2\rangle \mathbf{m}'_H$, with $H \cong H'$.

PROOF. It stems from the fact that b_1, b_2 (by definition of $\equiv$) are one the color-permutation of the other. ■

As explained (Procedure 1), the left-hand side L of an injective rule r is isomorphic to a graph whose nodes are $N_L = Var_I \cup Var_{obs}$ (the i^{th} node of L is identified by n_i), and whose edges are represented by the symbolic bag bag_{E_L}. Precisely, we define the edge set E_L as: $\forall\ T\ = \langle n_i, n_j, \mathrm{L}_k\rangle \in bag_{E_L}$, there are $bag_{E_L}(T)$ edges $e \in E_L$, such that $s(e) = n_i$, $t(e) = n_j$, $l(e) = \mathrm{L}_k$. The definition below meets this convention.

Definition 6 (match and corresponding instance) Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be an injective DPO rule, t_r its SN translation, G a graph, and $\mathbf{m}_G$ its encoding.

Given $m : L \to G$, we say that $b_m \in \mathcal{D}(t_r)$ is a *corresponding instance* if and only if $\forall n_i \in Var_I \cup Var_{obs}$: $b_m(n_i) = col_G(m_N(n_i))$, where col_G is the color-map used in $\mathbf{m}_G$. The class of corresponding instances is $[b_m]_{\equiv}$.

The other way round, let $b \in \mathcal{D}(t_r)$, such that $\mathrm{I}[\texttt{Node}, t_r](b) \leq \mathbf{m}_G[\texttt{Node}] \wedge \mathrm{I}[\texttt{Edge}, t_r](b) \leq \mathbf{m}_G[\texttt{Edge}]$. The *corresponding morphism* $m_b : L \to G$ is:

- $\forall n_i \in Var_I \cup Var_{obs} : m_{b_N}(n_i) = col_G^{-1}(n_i(b))$
- $\forall e \in E_L : m_{b_E}(e) = e' \in E_G$, with $s(e') = m_{b_N}(s(e))$, $t(e') = m_{b_N}(t(e))$, $l(e') = l(e)$

In the 2nd part of Definition 6, the assumption on the input functions ensures that the colors bound to any $n_i \in Var_I \cup Var_{obs}$ correspond to (describe) nodes of G (analogously for color-tuples describing edges). That is, the transition instance "matches" a graph morphism.

Definition 6 thus sets a bijection between morphisms matching an injective DPO rule r to a graph G and (equivalence classes of) instances of the transition t_r (the SN translation of r) verifying the input enabling condition in $\mathbf{m}_G$. For simplicity, symbol b_m will denote a representative of the class of instances corresponding to a given match m.

The following lemma precisely states the semantics of the SN translation of an injective DPO rule.

Lemma 2 *Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be an injective rule, t_r its SN translation, and G a graph.*

$m : L \to G$ satisfies the gluing condition (Property 4) if and only if (t_r, b_m) is enabled in $\mathbf{m}_G$.

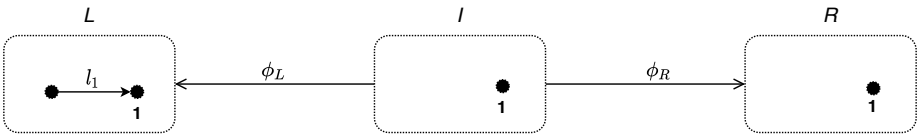


Figure 7: DPO rule removing a (source) node from a graph

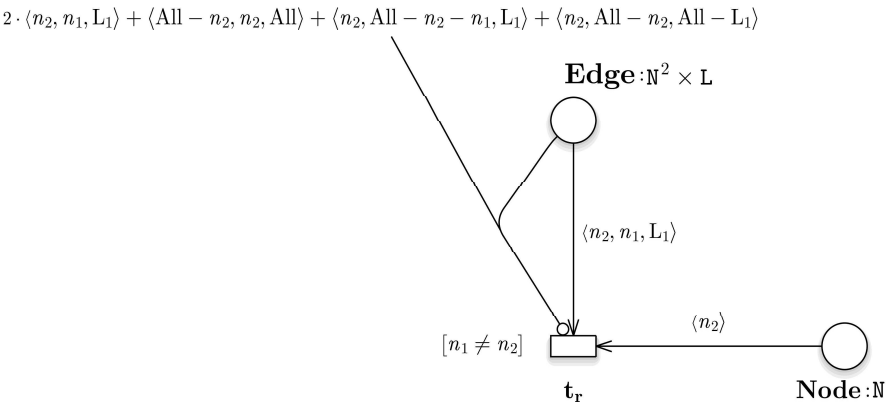


Figure 8: SN translation of rule in Figure 7

PROOF. Consider $\Rightarrow$. We assume that m is injective. Since $n_i(b_m) \neq n_j(b_m)$, $\forall n_i, n_j \in Var_I \cup Var_{obs}, i \neq j$ (i.e., n_i, n_j are bound to different colors), it holds $\mathrm{I}[\mathtt{Node}, t_r](b_m) \leq \mathbf{m}_G[\mathtt{Node}]$. By the way, $\mathrm{I}[\mathtt{Edge}, t_r](b_m) \leq \mathbf{m}_G[\mathtt{Edge}]$ because m is a morphism. We have just to consider inhibitor functions. Assume, by contradiction, that $\exists e$: $\mathrm{H}[\mathtt{Edge}, t_r](b_m)(e) \leq \mathbf{m}_G[\mathtt{Edge}](e)$; due to the definition of $\mathrm{H}[\mathtt{Edge}, t_r]$, this is only possible if there is $n_i \in Var_{obs}$ such that, letting $v_j = n_i(b_m)$ be the associated colour (obsolete node), edge e is incident to v_j and there are more instances of e than those which are withdrawn (better, required by the input arc function, whose image corresponds to the image of m). This would imply that there is an arc (an occurrence of e) incident to an obsolete node, which is not in the image of m_E, in contrast with the Hp (Property 4, dangling condition). As for $\mathrm{H}[\mathtt{Node}, t_r]$, we only have to chose colors for fresh variables such that $\forall n_i, n_j \in Var_{fresh}, i \neq j, \forall n_h \in Var_I \cup Var_{obs}$: $n_i(b_m) \neq n_j(b_m) \wedge n_i(b_m) \neq n_h(b_m) \wedge n_j(b_m) \neq n_h(b_m)$: this is surely possible, because we assume color class N large enough (all these instances are isomorphic).
Consider $\Leftarrow$. We have to show that, if (t_r, b) is enabled in $\mathbf{m}_G$, then m_b verifies Property 4. The identification condition is trivially met: since $\mathrm{I}[\mathtt{Node}, t_r](b) \leq \mathbf{m}_G[\mathtt{Node}]$, it holds: $n_i(b) \neq n_j(b)$, $\forall n_i, n_j \in Var_I \cup Var_{obs}, i \neq j$. As for the dangling condition, we can use pretty the same reasoning as for the 1st assertion. By definition of $\mathrm{H}[\mathtt{Edge}, t_r]$, we know that, if $n_i \in Var_{obs}$, $n_i(b) = v_j$, and a type-edge color tuple $e \in \mathbf{m}_G[\mathtt{Edge}]$ is "incident" to (contains) v_j, then there

are just as many occurrences of e in $\mathbf{m}_G[\texttt{Edge}]$ as those withdrawn by the firing of b ($\mathrm{I}[\texttt{Edge}, t_r](b)(e) = \mathbf{m}_G[\texttt{Edge}](e)$): that is, each edge of G incident to the image $m_b(v_j)$ of an obsolete node v_j of L is in turn in the image of m_b. ■

The parallelism between DPO rules and their SN translation is complete observing that, given a rule $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ and a match $m : L \to G$ satisfying the gluing condition, the application of r and the firing of any corresponding instance of rule's translation t_r have the same effect. This directly follows from the next explanation of a DPO rule's application, which is valid independently on whether m and ϕ_R are injective or not:

1. we remove the image of obsolete elements, $m(L - \phi_L(I))$, from G, to get the context graph C[10]; we directly derive morphism ν from m
2. we derive the target graph H by gluing C and R over I ($H \cong R +_{\phi_r, \nu} C$): we *merge* elements of R, C with a common pre-image in I, whereas we include in H all the other elements of R and C, individually taken.

If ϕ_R is injective we do not perform any merge. In such a case, if m is an injective match (we shall relax this condition) satisfying the gluing condition, and b_m is a representative of corresponding instances of t_r, then we can reproduce the two steps above in terms of SN firing rule (functions $\mathrm{W}^-[p,t], \mathrm{W}^+[p,t]$ are defined in Table 1). Let $p \in \{\texttt{Node}, \texttt{Edge}\}$:

1. $\mathbf{m}_G[p] - \mathrm{W}^-[p, t_r](b_m) = \mathbf{m}_C[p]$
2. $\mathbf{m}_C[p] + \mathrm{W}^+[p, t_r](b_m) = \mathbf{m}_H[p]$

Corollary 1 (of Lemma 2) *Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be an injective DPO rule, t_r its SN translation, and G a graph.*

$m : L \to G$ is a match satisfying the gluing condition such that $G \overset{r,m}{\Longrightarrow} H$ if and only if $\mathbf{m}_G[t_r, b_m\rangle \mathbf{m}_{G'}$, with $H \cong G'$.

7.3.1. Non-Injective Match

The translation of an injective DPO rule r into a single SN transition t_r, formalized by Procedure 1, only deals with injective matches of rule's left-hand side L to a host graph G. If we want to include non-injective matches, we have to modify a single point of the procedure. The definitions/outcomes presented in Section 7.3 remain valid. First, we deal with a possibly non-injective node map m_N, assuming the edge map m_E injective. Then, we exemplify the treatment of a non-injective m_E, which, however, has little interest in the practice.

A non-injective match m for the rule in Figure 5 identifies nodes 1,2 of L in G. The reason why the rule's translation (Figure 6), in either form (a) or (b), excludes such a match, is that the corresponding instance b_m of t_r assigns

[10] C is a graph, because the gluing condition ensures that the source and target nodes of an edge in C are nodes in C

variables n_1, $n_2 \in Var_I$ the same color of N (denoting the image in G of the two nodes of L). In the original translation, the instance (t_r, b_m) is not enabled because $\mathrm{I}[\texttt{Node}, t_r](b_m)$ is a non-type-set bag; in the equivalent form, (t'_r, b_m) is not even a valid transition instance.

The workaround to consider injective matches, however, is unexpectedly simple. It consists of writing the function(s) $\mathrm{I}[\texttt{Node}, t_r]$ ($\mathrm{O}[\texttt{Node}, t_r]$) as an elementary tuple with an inner summation, instead of a sum of elementary tuples, thus exploiting the "type-set" semantics of class-functions (Section 2.2.3).

Figure 9 shows an (injective) DPO rule with a non-injective match m and the corresponding graph transformation. Figure 10 shows the rule's alternative translations into SN: (a) comes directly from Procedure 1, i.e., it refers to injective matches of the rule, (b) refers, instead, to matches m with any m_N.

Passing from (a) to (b) requires two simple changes to color annotations: we erase variable n_3 ($Var^*_I = \{n_3\}$) from $\mathrm{I}[\texttt{Node}, t_r]$, $\mathrm{O}[\texttt{Node}, t_r]$ then we rewrite the residual expression of the input (output) arc-function as:

$$\langle n1 \rangle + \langle n_2 \rangle F \xrightarrow{\neq} \langle n_1 + n_2 \rangle$$

Due to the class-function semantics, the resulting expression is type-set and evaluates as the original one (only) if n_1, n_2 are bound (map) to different colors. This way, we encode non-injective morphisms $L \to G$ as t'_r instances where two or more variables among $\{n_1, n_2, n_3\}$ are bound (map) to the same color.

As for the rule in Figure 5, we get its SN translation including matches m with non-injective m_N by just erasing the transition's guard from the SN in Figure 6 (b).

Based on the above remarks, to deal with rule matches m with any m_N we have to slightly modify Step 2 of Procedure 1. Note that revised Step 2 embeds the reduction on I/O arc-functions specified by Property 7.

Procedure 1* (Procedure 1 with revised Step 2 - match m with any m_N)

$$\mathrm{I}[\texttt{Node}, t_r] = \langle \sum_{n_i \in Var_I^- \cup Var_{obs}} n_i \rangle \qquad \mathrm{O}[\texttt{Node}, t_r] = \langle \sum_{n_i \in Var_I^- \cup Var_{fresh}} n_i \rangle$$

($\mathrm{H}[\texttt{Node}, t_r]$ is unchanged)

Is is straightforward to find out the implicit predicates which characterize the enabled instances of the SN translation of a DPO rule r including also non-injective matches of r.

Property 9 Let t_r be the SN transition obtained by translating an injective DPO rule $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ according to Procedure 1* and $\mathbf{m}$ a graph-encoding.

If (t_r, b) is enabled in $\mathbf{m}$ then

$$\forall n_i \in Var(t_r)\ \forall n_j \in Var(t_r) - Var_I,\ i \neq j :\ n_i(b) \neq n_j(b)$$

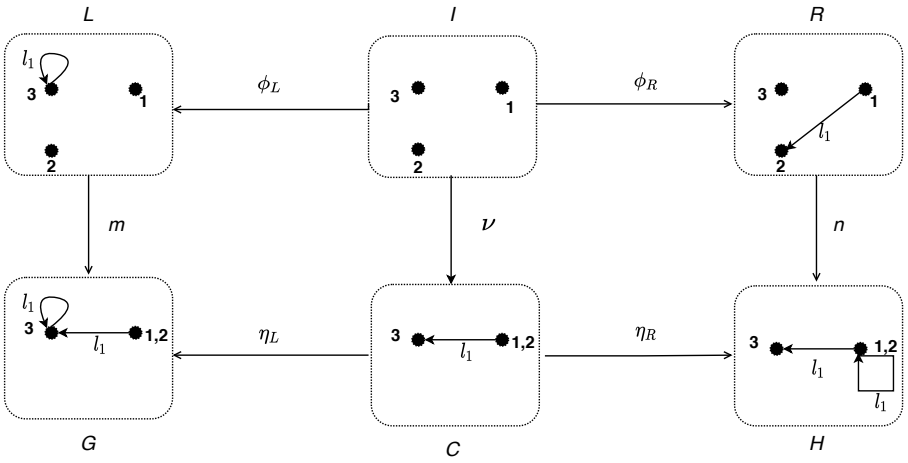


Figure 9: DPO rule with non-injective match m_N

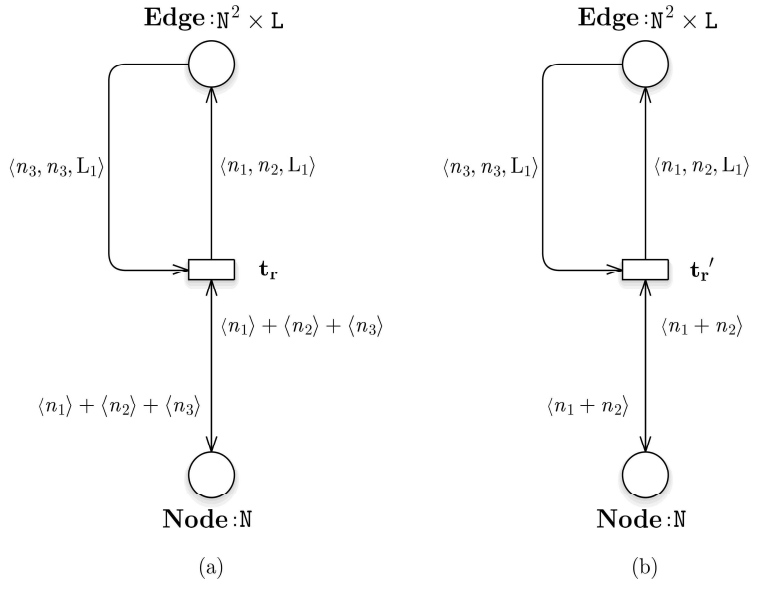


Figure 10: SN translations of rule in Figure 9: injective m_N vs any m_N

A very important thing is that all the notions/outcomes presented in Section 7.2 still hold, except Property 6 (replaced by Property 9). This statement has two main explanations. As for the gluing condition, the few changes introduced to the translation procedure allow a valid instance of t_r (corresponding to a non-injective match m) to identify only variables denoting non-obsolete nodes. As for the dangling-edge condition, the changes above do not even involve it. The second (related) reason lies in the new shape of the function $\mathrm{I}[\texttt{Node}, t_r]$: this function is type-set, hence, also instances corresponding to non-injective morphisms may be enabled in a graph-encoding.

Non-injective m_E. Figure 11 shows a DPO rule whose associated match m is such that m_E is non-injective (m_E identifies the two edges with label lb_1 of L in the host graph G). Figure 12 shows two alternative SN translations of the rule, one (a) obtained according to Procedure 1* (the functions on I/O arcs incident to place `Node` are null because $Var_I^- = \emptyset$), the other (b) taking account of non-injective edge-matches m_E. The arc-function $\mathrm{I}[\texttt{Edge}, t_r]$ contains the sub-sum $F = \langle n_1, n_3, \mathrm{L}_1 \rangle + \langle n_1, n_2, \mathrm{L}_1 \rangle$, whose two terms represent edges of L which may be identified by a morphism m_E, prior identification of their target nodes. Therefore, we rewrite the subsum into a (non-equivalent) *disjoint* form:

$$F \xrightarrow{\not\equiv} F[n_2 \neq n_3] + \langle n_1, n_2, \mathrm{L}_1 \rangle [n_2 = n_3] \equiv \langle n_1, n_2 + n_3, \mathrm{L}_1 \rangle$$

whose elements represent injective/non-injective m_E, respectively. We can express this form compactly, utilizing the class-function $n_2 + n_3$.

The general formalization is merely a technical matter: for each (maximal) subset of edges (symbolic tuples) $\{\langle n_i, n_j, \mathrm{L}_k \rangle\}$ of graph L with an identical label[11], we should consider all its partitions, and encode each of these (modelling any edge map) through a set of (in)equalities associated to a term of $\mathrm{I}[\texttt{Edge}, t_r]$ (we get mutually exclusive guards). Let us omit the details of this operation (combinatorial with respect to $|bag_{E_L}|$), which is of little interest, both theoretical and practical. Even though non-injective edge-matches of a rule exhibit a kind of "folding" effect, they ultimately obscure rule's interpretation.

7.4. Non-Injective DPO Rules

Until now, we have assumed that both legs of a rule $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ are injective. This is actually a general assumption. Finally, we consider the case of a non-injective right leg ϕ_R, what may result in *merging* nodes/edges of a source graph G.

A non-injective rule and related graph transformation are shown in Figure 13. The nodes 2 and 3 of the interface I, in fact, are identified by ϕ_R, together with the two incident arcs. The rule's application to G, via the match m, results in $H = R +_I C$: elements of R and C with a common pre-image in I are merged. As a result, nodes 2 and 3 of G are merged, as well as two of the three edges

[11] identification of parallel edges of graph L is coherently excluded

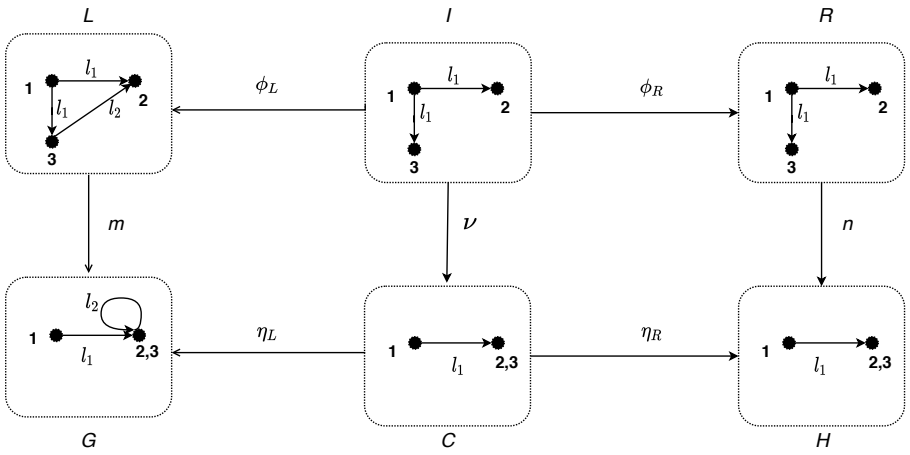


Figure 11: DPO rule with a non-injective match m_E and graph transformation

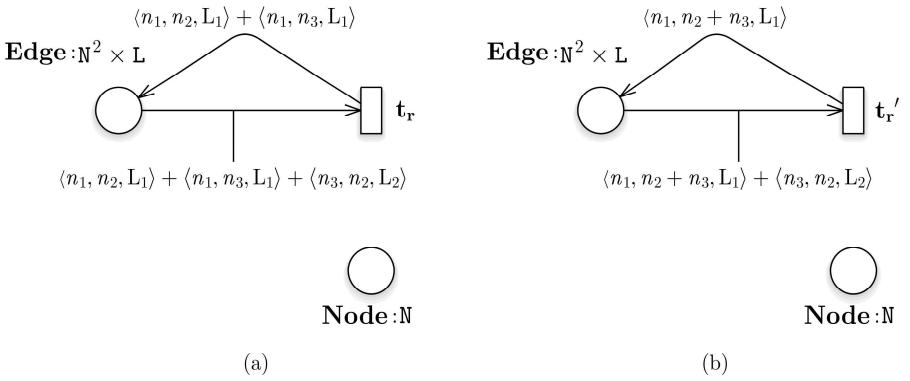


Figure 12: SN translations of DPO rule in Figure 11: injective m_E (any m_N) vs *any* m_E

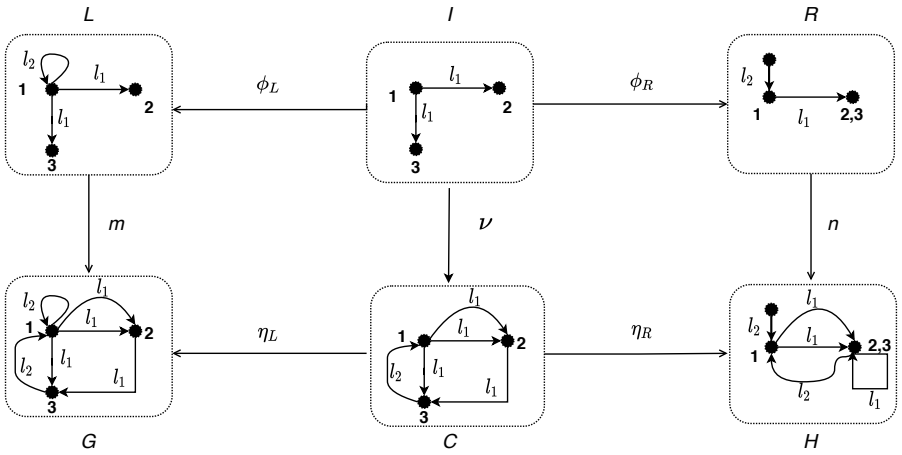


Figure 13: a NON-injective DPO rule and a graph transformation

(with label lb_1) departing from node 1. The edges incident to nodes 2,3 of G are "redirected" in H to the node resulting from merge.

Being able to reproduce this context-dependent changes, preserving the atomic semantics of DPO rules, requires the use of a SN *composite* rule (Section 6). The subnet's *observable* transition, t_r, directly implements most of the structural changes on a source graph G (merge/add/removal of elements), and triggers a sequence of *immediate* transitions which rebuild the adjacencies of merged nodes on the target graph H. The color annotations of t_r are slightly different from the injective rule case, the rest of the subnet is built according to a structural pattern.

Figure 14 shows the SN subnet translating the rule in 13. Let us formally describe the translation procedure, focusing on new elements.

Procedure 2 (SN translation of non-injective DPO rules - any match m) Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be a DPO rule, with ϕ_R non-injective and ϕ_L injective. We refer to any match m, with m_E injective.

1. The rule's observable transition, t_r, is linked to places `Node`, `Edge` as described in Section 7.2, with a few differences:
   - $Var_{fresh} \cong (N_R - \phi_R(N_I)) \cup \{v \in N_r, |\phi_R^{-1}(v)| > 1\}$. In other words, the nodes of R with non-singleton pre-images are described as *fresh* variables, denoted Var_{merge}.
   - I[`Node`, t_r], O[`Node`, t_r] are defined as in Step 2 of Procedure 1*.
   - $\forall n_i \in Var_{merge}\ \forall n_j, n_k \in \phi_R^{-1}(n_i), j \neq k$, there is $n_j \neq n_k$ in $g(t_r)$

   (Note that in general $bag_{E_I} \not\leq bag_{E_R}$)
2. for each $n_i \in Var_{merge}$, there are a pair of places $\{\texttt{ToMerge}_i, \texttt{Merge}_i\}$ and a pair transitions $R_i = \{\texttt{reconnect\_s}_i, \texttt{reconnect\_t}_i\}$, such that, letting

x, y, be auxiliary, type-N variables, and t_i denote any R_i element:

$$\mathrm{O}[\texttt{ToMerge}_i, t_r] = \mathrm{I}[\texttt{ToMerge}_i, t_i] = \mathrm{O}[\texttt{ToMerge}_i, t_i] = \mathrm{I}[\texttt{ToMerge}_i, end] = \sum_{n_j \in \phi_R^{-1}(n_i)} \langle n_j \rangle$$

$$\mathrm{O}[\texttt{Merge}_i, t_r] = \mathrm{I}[\texttt{Merge}_i, t_i] = \mathrm{O}[\texttt{Merge}_i, t_i] = \mathrm{I}[\texttt{Merge}_i, end] = \langle n_i \rangle$$

$$\mathrm{I}[\texttt{Edge}, t_i] = \langle x, y, l \rangle \qquad \mathrm{O}[\texttt{Edge}, \texttt{reconnect\_s}_i] = \langle n_i, y, l \rangle$$
$$\mathrm{O}[\texttt{Edge}, \texttt{reconnect\_t}_i] = \langle x, n_i, l \rangle$$

$$g(\texttt{reconnect\_s}_i) = \bigvee_{n_j \in \phi_R^{-1}(n_i)} x = n_j \quad g(\texttt{reconnect\_t}_i) = \bigvee_{n_j \in \phi_R^{-1}(n_i)} y = n_j$$

3. transition *end* is linked to place `Edge` through an inhibitor arc

$$\mathrm{H}[\texttt{Edge}_i, end] = \langle F, All, All \rangle + \langle All - F, F, All \rangle \qquad \text{where}$$
$$F = \sum_{n_i \in Var_{merge}, n_j \in \phi_R^{-1}(n_i)} \langle n_j \rangle$$

4. Places `Start`,`Trigger` are linked to the sub-net's transitions, as indicated at (the first two points of) Definition 4

The observable transition, t_r, manages the possible identification of edges by ϕ_R, and the other rewritings involving obsolete/fresh graph elements (if any). The couple of transitions $\texttt{reconnect\_s}_i$, $\texttt{reconnect\_t}_i$ manage the redirection of edges incident to any of the (source/target) nodes which are merged into a fresh new one. As a last step, transition *end* removes all the merged nodes, according with the dangling-edge condition.

Since, by construction, all the transitions of the SN translating a non-injective DPO rule match Definition 2 (and the other conditions set by Definition 4), it holds:

Property 10 Let r be a non-injective DPO rule. The SN subnet obtained from r according to Procedure 2 is a SN composite GTR.

As for the example in Figure 13, we get: $Var_I = \{n_1, n_2, n_3\}$ ($Var_I^- = \emptyset$), $Var_{fresh} = \{n_4, n_5\}$, $Var_{merge} = \{n_4\}$, $Var_{obs} = \emptyset$, $\phi_R^{-1}(n_4) = \{n_2, n_3\}$. The arc-functions and the guard annotating t_r turn out to be:

$$\mathrm{I}[\texttt{Node}, t_r] = \epsilon \qquad \mathrm{O}[\texttt{Node}, t_r] = \mathrm{H}[\texttt{Node}, t_r] = \langle n_4 + n_5 \rangle$$
$$\mathrm{I}[\texttt{Edge}, t_r] = \langle n_1, n_1, \mathrm{L}_2 \rangle + \langle n_1, n_2, \mathrm{L}_1 \rangle + \langle n_1, n_3, \mathrm{L}_1 \rangle$$
$$\mathrm{O}[\texttt{Edge}, t_r] = \langle n_5, n_1, \mathrm{L}_2 \rangle + \langle n_1, n_4, \mathrm{L}_1 \rangle \qquad \mathrm{H}[\texttt{Edge}, t_r] = \epsilon$$
$$g(t_r) = n_2 \neq n_3 \wedge n_4 \neq n_5$$

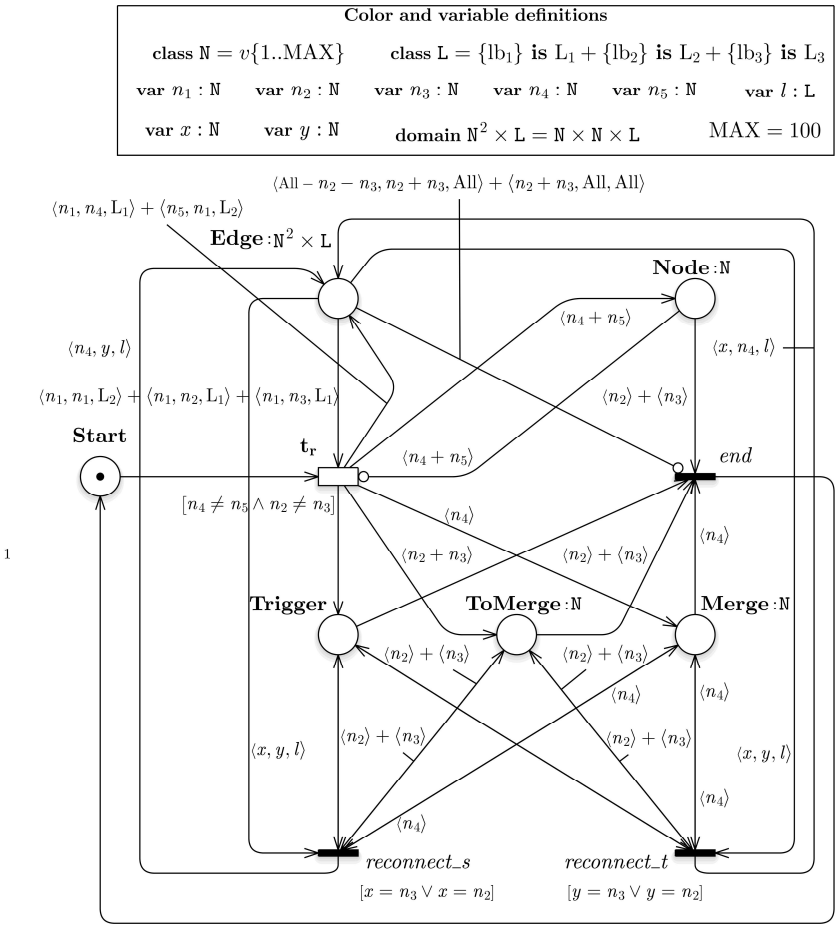


Figure 14: SN translation of rule in Figure 13

The inequality $n_2 \neq n_3$ in the guard of t_r (Step 1 of Procedure 2, last point) makes the effect of a non-injective match m not to overlap with that of the non-injective morphism ϕ_R, which identifies nodes 2,3. Such an overlap would result in an undefined, or at least obscure, behaviour.

Accordingly, we need to modify (a bit) the match notion for non-injective DPO rules.

Definition 7 (match of a non-injective rule) Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be a rule with ϕ_R non-injective and ϕ_L injective, and G be a graph.
A morphism $m : L \to G$ is a *match* if and only if m_E is injective and
$\forall v_i \in N_R, |\phi_R^{-1}(v_i)| > 1 \; \forall v_j, v_k \in \phi_L(\phi_R^{-1}(v_i)), j \neq k : m_N(v_j) \neq m_N(v_k)$

We may thus easily extend to non-injective rules the notion of transition instance(s) corresponding to a match (Definition 6), by referring to the observable transition t_r of the SN subnet translating one such rule.

We have to rephrase Corollary 1 by taking into account the fact that the observable transition triggers a sequence of immediate transitions.

Corollary 2 *Let $r = L \xleftarrow{\phi_L} I \xrightarrow{\phi_R} R$ be a non-injective rule, t_r the observable transition of the subnet obtained from r according to Procedure 2, G a graph. If $m : L \to G$ satisfies the gluing condition and $G \overset{r,m}{\Longrightarrow} H$ then $\mathbf{m}_G[t_r, b_m \cdot \rho\rangle \mathbf{m}_{G'}$, where ρ is a firing sequence of immediate transitions terminated by transition end and $H \cong G'$. And vice versa.*

A final note concerns the left leg ϕ_L of a rule. Almost all papers on graph transformation assume that it is injective. The reason is that, if ϕ_L is not injective, there may be cases when, even if a context C does exist for a match m of L in a host graph G, it is not unique (for m). That may result in non-determinism, because some nodes of G may have to be *split*, and the incident edges may decide to "follow" any of split nodes. This situation shows very little practical interest. However, we conjecture it might be represented without any difficulty using composite rules, being in some sense dual to the case of a non-injective right leg ϕ_R.

7.5. From SN GTR to DPO Rules

We conclude the comparison between SN Graph Transformation Rules and DPO rules by shortly discussing the opposite direction, going from SN to DPO. In our opinion, it is much less intriguing. The main reason is that there are lots of rule application conditions, especially of restrictive type, that we can easily model in SN, instead we hardly, or even cannot, embed in a (base) DPO approach (or other algebraic ones.) The typical case consists of inhibiting conditions on the left-hand side of a rule, e.g., when computing graph transitive closure (whenever there exists an edge from s to v and one from v to t, create one from s to t only if such an edge is absent). Other conditions we can easily set with SN (and cannot so easily with DPO) concern edge labels (e.g., removing

edges with labels different from a given one), multiplicity (e.g, removing parallel instances of an edge, leaving a single one), and so forth.

We believe that it is possible to characterize a quite large subclass of SN GTR (especially of elementary type) that can be mechanically translated into corresponding DPO rules. This might be done by defining syntactical patterns for arc-functions that are likely matched by most practical examples: the SN rules shown in the gallery of Figure 1 (but R5), e.g., are easily translatable into DPO rules. This is part, however, of ongoing work.

8. Conclusions and Ongoing Work

In this paper, we have significantly enhanced and extended the outcome of a recent one [18] which, reversing the usual perspective, has proposed a formalization of Graph Transformation Systems (GTS) in terms of Symmetric Nets (SN). SN are a standard, High-level PN formalism featuring a compact syntax highlighting system behavioural symmetries, which are exploited in efficient analysis techniques. We use a structural approach, exploiting a recently implemented symbolic calculus for SN, to validate Graph Transformation Rules and formally verify properties of GTS in an efficient way. A major strength is given by the possibility of using automated tools for the editing and analysis of graph transformation models.

In the second part of the paper, we have provided a new semantic characterization of the SN-based graph transformation approach, by carrying out a thorough comparison with the classical, algebraic double-pushout (DPO) approach. We have shown in a constructive way that any DPO rule maps to a corresponding, elementary or composite, SN rule. We have treated both injective and non-injective DPO rules and rule matches. The comparison shows that the SN approach is, in some sense, a generalization of the DPO.

Throughout the paper, we have used simple but significant examples of graph encoding through SN. Edge-labelled, multi-graphs are considered.

Ongoing/Future Work. We are extending the SN-based graph transformation by encoding Place/Transitions (P/T) nets enriched with inhibitor arcs, a kind of directed, bipartite graphs. A SN emulator for P/T nets (encoded as markings) has been defined in [30], together with a simple API for base net-transformations. We aim at integrating the SN emulator with the theoretically sound, rule-based graph transformation approach presented in this paper.

More general classes of graphs, in particular hypergraphs, cannot simply encoded with SN, due to the SN low data abstraction ability. On the other side, for the same reason, graph transformations with a richer (typed, constrained) labelling mechanism, generally known as Attributed Graph Rewriting [27, 28], are hardly translatable into SN. To cover more complex graph encodings and graph transformation approaches, Algebraic (Spec-inscribed) Petri nets [20] may be more conveniently used [31]. Unfortunately, this formalism pays its higher expressivity with reduced analysis capability and automated support, so its usage requires further studies.

References

[1] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, WORLD SCIENTIFIC, 1997. doi:10.1142/3303.

[2] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages, and Tools, WORLD SCIENTIFIC, 1999. doi:10.1142/4180.

[3] H. Ehrig, M. Pfender, H. J. Schneider, Graph-Grammars: An Algebraic Approach, in: Proceedings of the 14th Annual Symposium on Switching and Automata Theory (Swat 1973), SWAT '73, IEEE Computer Society, USA, 1973, p. 167–180. doi:10.1109/SWAT.1973.11.

[4] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe, Algebraic approaches to graph transformation – part i: Basic concepts and double pushout approach, in: Rozenberg [1], pp. 163–245. doi:10.1142/9789812384720_0003.

[5] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of algebraic graph transformation., Berlin: Springer, 2006. doi:10.1007/3-540-31188-2.

[6] V. Danos, J. Feret, W. Fontana, R. Harmer, J. Hayman, J. Krivine, C. Thompson-Walsh, G. Winskel, Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models, in: S. D. L.-Z. fuer Informatik (Ed.), FSTTCS 2012 - IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, Vol. 18 of LIPIcs, Hyderabad, India, 2012, pp. 276–288. doi:10.4230/LIPIcs.FSTTCS.2012.276.

[7] M. Fernández, H. Kirchner, B. Pinaud, J. Vallet, Labelled graph rewriting meets social networks, in: D. Lucanu (Ed.), Rewriting Logic and Its Applications, Springer International Publishing, Cham, 2016, pp. 1–25. doi:10.1007/978-3-319-44802-2_1.

[8] K. Do, T. Tran, S. Venkatesh, Graph transformation policy network for chemical reaction prediction, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 750–760. doi:10.1145/3292500.3330958.

[9] W. Reisig, Petri Nets: An Introduction, Springer-Verlag New York, Inc., New York, NY, USA, 1985. doi:10.1007/978-3-642-69968-9.

[10] W. Fokkink, Introduction to process algebra., Berlin: Springer, 2000.

[11] D. Sangiorgi, D. Walker, The π-calculus: A theory of mobile processes., Cambridge: Cambridge University Press, 2001.

[12] H. Ehrig, B. König, Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts., Math. Struct. Comput. Sci. 16 (6) (2006) 1133–1163.

[13] J. Padberg, L. Kahloul, Overview of Reconfigurable Petri Nets, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, Springer International Publishing, Cham, 2018, pp. 201–222. doi:10.1007/978-3-319-75396-6_11.

[14] H.-J. Kreowski, A comparison between Petri-nets and graph grammars, in: H. Noltemeier (Ed.), Graphtheoretic Concepts in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 1981, pp. 306–317.

[15] A. Corradini, Concurrent graph and term graph rewriting, in: U. Montanari, V. Sassone (Eds.), CONCUR '96: Concurrency Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 438–464. doi:10.1007/3-540-61604-7_69.

[16] P. Baldan, A. Corradini, F. Gadducci, U. Montanari, From Petri Nets to Graph Transformation Systems, ECEASST 26 (01 2010). doi:10.14279/tuj.eceasst.26.368.

[17] H. Ehrig, J. Padberg, Graph Grammars and Petri Net Transformations, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets: Advances in Petri Nets, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 496–536. doi:10.1007/978-3-540-27755-2_14.

[18] L. Capra, An operational semantics of graph transformation systems using symmetric nets, in: M. Marin, A. Crăciun (Eds.), Proceedings Third Symposium on Working Formal Methods, Timişoara, Romania, 3-5 September 2019, Vol. 303 of Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2019, pp. 107–119. doi:10.4204/EPTCS.303.8.

[19] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, Stochastic well-formed colored nets and symmetric modeling applications, IEEE Transactions on Computers 42 (11) (1993) 1343–1360. doi:10.1109/12.247838.

[20] K. Jensen, G. Rozenberg (Eds.), High-level Petri Nets: Theory and Application, Springer-Verlag, London, UK, 1991. doi:10.1007/978-3-642-84524-6.

[21] K. Jensen, Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use., Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing 1997. ISBN: 3-540-60943-1., 1997. doi:10.1007/978-3-662-03241-1.

[22] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, G. Franceschinis, 30 years of GreatSPN, in: Principles of Performance and Reliability Modeling and Evaluation, Springer, 2016, pp. 227–254. doi:10.1007/978-3-319-30599-8_9.

[23] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, A symbolic reachability graph for coloured Petri nets, Theoretical Computer Science 176 (1) (1997) 39 – 65. doi:10.1016/S0304-3975(96)00010-2.

[24] L. Capra, M. D. Pierro, G. Franceschinis, A High Level Language for Structural Relations in Well-Formed Nets, in: Proc. of the 26th Int. Conf. ATPN 2005, Vol. LNCS 3536, Springer, 2005, pp. 168–187. doi:10.1007/11494744_11.

[25] L. Capra, M. De Pierro, G. Franceschinis, Computing Structural Properties of Symmetric Nets, in: J. Campos, B. R. Haverkort (Eds.), Quantitative Evaluation of Systems QEST 2015, Springer International Publishing, Cham, 2015, pp. 125–140. doi:10.1007/978-3-319-22264-6_9.

[26] L. Capra, M. De Pierro, G. Franceschinis, SNexpression: A Symbolic Calculator for Symmetric Net Expressions, in: R. Janicki, N. Sidorova, T. Chatain (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2020, pp. 381–391. doi:10.1007/978-3-030-51831-8_19.

[27] B. König, V. Kozioura, Towards the verification of attributed graph transformation systems, in: H. Ehrig, R. Heckel, G. Rozenberg, G. Taentzer (Eds.), Graph Transformations, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 305–320. doi:10.1007/978-3-540-87405-8_21.

[28] F. Orejas, Symbolic graphs for attributed graph constraints., J. Symb. Comput. 46 (3) (2011) 294–315.

[29] M. Löwe, Algebraic approach to single-pushout graph transformation., Theor. Comput. Sci. 109 (1-2) (1993) 181–224.

[30] L. Capra, M. Camilli, Towards Evolving Petri Nets: a Symmetric Nets-based Framework, IFAC-PapersOnLine 51 (7) (2018) 480 – 485, 14th IFAC Workshop on Discrete Event Systems WODES 2018. doi:10.1016/j.ifacol.2018.06.343.

[31] L. Capra, A pure SPEC-inscribed PN model for reconfigurable systems, in: 2016 13th International Workshop on Discrete Event Systems (WODES), IEEE, 2016, pp. 459–465. doi:10.1109/WODES.2016.7497888.

[32] L. Capra, M. D. Pierro, G. Franceschinis, A Tool for Symbolic Manipulation of Arc Functions in Symmetric Net Models, in: Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools, ValueTools '13, ICST, Torino, Italy, 2013, pp. 320–323. doi:10.4108/icst.valuetools.2013.254407.

[33] C. Dutheillet, S. Haddad, Conflict Sets in Colored Petri Nets, in: proc. of Petri Nets and Performance Models, 1993, pp. 76–85. doi:10.1109/PNPM.1993.393433.