A Framework for the Verification of Infinite-State Graph Transformation Systems [1]

Paolo Baldan [a,*], Andrea Corradini [b], Barbara König [c],

[a] *Dipartimento di Matematica Pura e Applicata, Università di Padova, Italia*

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

[c] *Abt. für Informatik und Ang. Kognitionswissenschaft, Universität Duisburg-Essen, Germany*

Abstract

We propose a technique for the analysis of infinite-state graph transformation systems, based on the construction of finite structures approximating their behaviour. Following a classical approach, one can construct a chain of finite under-approximations (*k-truncations*) of the Winskel style unfolding of a graph grammar. More interestingly, also a chain of finite over-approximations (*k-coverings*) of the unfolding can be constructed. The fact that k-truncations and k-coverings approximate the unfolding with arbitrary accuracy is formalised by showing that both chains converge (in a categorical sense) to the full unfolding. We discuss how the finite over- and under-approximations can be used to check properties of systems modelled by graph transformation systems, illustrating this with some small examples. We also describe the AUGUR tool, which provides a partial implementation of the proposed constructions, and has been used for the verification of larger case studies.

Key words: Graph transformation, Petri nets, category theory, abstraction, verification.
2000 MSC: 68Q42, 68Q60, 68Q55

* Corresponding author.
Email addresses: baldan@math.unipd.it (Paolo Baldan), andrea@di.unipi.it (Andrea Corradini), barbara_koenig@uni-due.de (Barbara König).

[1] Research partially supported by the EC RTN 2-2001-00346 Project SEGRAVIS, the MIUR Project ART, the DFG project SANDS, Programma Vigoni (CRUI/DAAD) "Models based on Graph Transformation Systems: Analysis and Verification", and project SENSORIA, IST-2005-016004.

1 Introduction

With the advent of mobile and ubiquitous computing, modern software and computer systems are frequently characterised by a high level of dynamicity. Features such as flexible topologies, the dynamic creation and deletion of objects, and an infinite-state space make them very hard to analyse and verify.

In this context, *graph transformation systems* (GTSs) [52] emerge as a powerful specification formalism for concurrent, distributed and mobile systems [23], generalising another classical model of concurrency, namely Petri nets [46]. For instance, graphs can be used to represent the logical and topological relations among the components of a distributed system, the connectivity in a network, the rights that system entities have over resources, the structure of the heap for a program with dynamic pointer structures. For highly dynamic systems, where, e.g., changes in the connectivity or in the structure of the network are part of the normal behaviour, the dynamics of the system can be naturally expressed by means of graph rewriting rules. Graph transformation systems can be used as a specification language in themselves (see, e.g., [23]), or as a kind of meta-language where other formalisms and languages for concurrency, e.g., process calculi, can be encoded (see, e.g., [26,43]).

Along the years the concurrent behaviour of GTSs has been deeply studied and a consolidated theory of concurrency is now available [52,23]. In particular, by exploiting the relationship with Petri nets, several concurrent semantics developed for nets, like process and unfolding semantics, have been extended to GTSs (see, e.g., [16,51,8,9]). However, concerning automated verification, which is crucial in the analysis of dynamically evolving systems, the rich literature on GTSs contains just a few contributions dealing with the static analysis of such systems (see [32,27,29,57,47] and the remarks about related work in Section 6).

Instead, several approaches have been successfully proposed for the analysis of Petri nets, ranging from the calculus of invariants [46] to model checking based on finite complete prefixes [41,25]. Some of such approaches, most notably the one originally proposed by McMillan in [41], are based on the concurrent semantics of nets, and more precisely on their *unfolding semantics*. This allows to avoid the combinatorial explosion arising when one explores all possible interleavings of concurrent events, thus contributing to alleviate the state explosion problem typical in the analysis of concurrent systems. Briefly, the unfolding of a Petri net is a single structure which fully describes the concurrent behaviour of the given system, including all possible transition occurrences and their mutual dependencies, as well as all reachable markings. In general, the unfolding is an infinite structure for any non-trivial net, but it has been shown that if the net is *bounded* (i.e., the set of reachable markings is finite), then it is possible to construct a finite, initial part of the unfolding, called *finite*

complete prefix, which provides as much information as the full unfolding.

From these considerations, a natural question arises: By exploiting the relationship between nets and graph transformation systems, is it possible to devise automated verification techniques for GTSs which exploit their concurrent semantics? This question has been answered positively recently for *finite state* GTSs in [6], where a first contribution to a theory of finite complete prefixes for such systems has been presented.

In the present paper, by elaborating and generalising the work presented in [4,10], we go further in this direction, presenting the foundations of a methodology for verifying *infinite-state* graph transformation systems. A common pattern used in the literature for verifying infinite-state systems, consists of considering an abstraction $\mathcal{A}$ of a concrete semantical model, providing a simpler description of the behaviour of the original system. Such a description is approximative, but still useful to check some properties of interest. More specifically, a class $\mathcal{L}$ of properties of interest is singled out, such that given any property φ in $\mathcal{L}$, the validity of φ in the abstraction $\mathcal{A}$ implies its validity in the original system. In some optimal cases, also the converse holds, i.e., the abstraction is "exact" for the properties in $\mathcal{L}$.

In this paper we follow this pattern, by providing a characterisation of several finite approximations of the full unfolding of GTSs, and showing how they can be used for verification. The approach is constructive, and a prototypical tool for the construction of such approximations has been implemented, as discussed in Section 5.3. As in the case of Petri nets, the full unfolding of a GTS is a structure which fully describes the concurrent behaviour of the system, including all possible rewriting steps and their mutual dependencies, as well as all reachable states [51,9]. Given a *graph grammar*, i.e., a GTS equipped with a start hypergraph, we show how to construct finite approximations of the full unfolding of the grammar, at any chosen level k of accuracy. The approximations can be arbitrarily close to the real behaviour of the systems, in a way that the corresponding chain of (both under- and over-) approximations converges to the exact behaviour.

More specifically, we will approximate GTSs by Petri nets, a conceptually simpler formalism which shares with the GTS model several interesting properties, such as locality (state changes are only described locally) and concurrency (no unnecessary interleaving of events), and for which several verification techniques have already been developed.

In more detail, in the paper we will consider the following two kinds of approximations:

Under-approximations (k-truncations). The unfolding of a graph grammar $\mathcal{G}$ can be defined as the union (categorically, the colimit) of its prefixes of finite causal depth. Hence "under-approximations" of the behaviour of $\mathcal{G}$ can be easily produced by stopping the construction of the unfolding at a finite causal depth k, thus obtaining the so-called *k-truncation* $\mathcal{T}^k(\mathcal{G})$ of the unfolding of $\mathcal{G}$. In the case of Petri nets this is at the basis of the finite prefix approach mentioned above: if the system is finite state and if the stop condition is suitably chosen, the prefix turns out to be complete, i.e., it contains the same information as the full unfolding [41,25]. In general, for infinite-state systems, any truncation of the unfolding will be just an under-approximation of the behaviour of the system, in the sense that any computation in the truncation can be executed in the original system as well, but not vice versa. Nevertheless, finite truncations can still be used to check interesting properties of the grammar, e.g., some liveness properties of the form "eventually A" for a predicate A (see Section 5.1).

Over-approximations (k-coverings). A more challenging issue is to provide sensible over-approximations of the behaviour of a grammar $\mathcal{G}$, i.e., finite approximations of the unfolding which "represent" all computations of the original system, but possibly more. To this aim, we propose an algorithm which, given a graph grammar $\mathcal{G}$, produces a finite structure, called *Petri graph*, consisting of a hypergraph and of a P/T net (possibly not safe, and potentially cyclic) over it, which can be seen as an (over-)approximation of the unfolding. The outcome of the algorithm is not uniquely determined by the graph grammar, but changes according to the chosen level of accuracy: essentially one can require the approximation to be exact up to a certain causal depth k, thus obtaining the so-called *k-covering* $\mathcal{C}^k(\mathcal{G})$ of the unfolding of $\mathcal{G}$.

The covering $\mathcal{C}^k(\mathcal{G})$ over-approximates the behaviour of $\mathcal{G}$ in the sense that every computation in $\mathcal{G}$ is mapped to a valid computation in $\mathcal{C}^k(\mathcal{G})$ and every hypergraph reachable from the start graph can be mapped homomorphically to (the graphical component of) $\mathcal{C}^k(\mathcal{G})$, and its image is reachable in the Petri graph. This allows us to identify a suitable class of graph properties (those reflected by graph morphisms) such that, if they hold for all graphs reachable in the covering $\mathcal{C}^k(\mathcal{G})$ then they also hold for all reachable graphs in $\mathcal{G}$. Important properties of this kind are the non-existence or non-adjacency of edges with specific labels, the absence of certain paths (which could be used for checking security properties) or cycles (for checking deadlock-freedom). Temporal properties, such as several safety properties of the form "always A", can be proved directly on the Petri net component of the coverings (see Section 5.1).

The theory is developed in this paper for graph transformation systems defined according to the double-pushout (DPO) approach [22], where nodes cannot be deleted. Note that, for modelling purposes the deletion of a node can often be

simulated by leaving it isolated, as we shall discuss in Section 2.1. Preliminary results were presented in [4,10], where additional restrictions were imposed on rules:

- rewriting rules could not check for the presence of edges which were not deleted (formally, the interface graph was discrete). Lifting this restriction does not make the formalism more expressive (since the preservation of edges can be simulated by deleting and recreating edges), but avoids an unnecessary loss of concurrency in the approximations.
- no pair of edges in the left-hand side graph of a rule could have the same label.

These two restrictions allowed a simpler technical treatment in the referred papers, because the Petri net component of a Petri graph was a standard Place/Transition net. Here, for the sake of greater generality, we shall use a more elaborated model of nets. More precisely, in order to handle rules with a possibly non-discrete interface (modelling read-only access to edges), we shall use *contextual Petri nets*, i.e., Petri nets enriched with read arcs [44,58,30], as the net component of a Petri graph. Furthermore, in order to allow for multiple edges with the same label in the left-hand side of a rule, it is technically convenient to resort to a variation of nets called *pre-nets* [13]. In a pre-net, a total ordering is imposed on the places occurring in the pre- and post-set of transitions.

The rest of the paper is structured as follows. In Section 2 we introduce the class of (hyper)graph transformation systems we will deal with, some basics of (contextual) Petri nets and the notion of pre-net. Then we present the notion of Petri graph, the structure used to represent and approximate the behaviour of GTSs. In Section 3 we define the k-truncations of the unfolding of a grammar, and the full unfolding itself as colimit of the truncations. In Section 4 we introduce the k-coverings of the unfolding, proving their main properties. In particular, the main result of this section shows that the algorithm computing k-coverings is correct, terminating and confluent. Furthermore we prove that the full unfolding is the categorical limit of the chain of the k-coverings. We discuss applications, some simple examples of mobile systems modelled as GTSs and the tool AUGUR in Section 5. Finally, in Section 6 we draw some conclusions and indicate directions of further investigations.

In this paper we will use basic notions from category theory in order to describe some concepts in a concise way and in order to simplify the proofs. Specifically we are using the notions of limit and colimit. For an introduction see [1,40].

2 Hypergraph rewriting, Petri nets and Petri graphs

In this section we first present the class of (hyper)graph transformation systems considered in the paper. Then, after recalling some basic notions about Petri nets, we will introduce Petri graphs, the structures combining hypergraphs and Petri nets which will be used to represent the (approximations of the) behaviour of GTSs.

2.1 Graph transformation systems

In the following, given a set A we denote by A^* the set of finite sequences of elements of A (i.e., the elements of the free monoid over A). Given $u \in A^*$ we write $|u|$ to indicate the length of u and $[u]_i$ to denote the i-th element of u. Furthermore, if $f : A \to B$ is a function then we denote by $f^* : A^* \to B^*$ its extension to sequences. Throughout the paper Λ denotes a fixed set of *labels*, where each label $l \in \Lambda$ is associated with an *arity* $ar(l) \in \mathbb{N}$.

Definition 1 (hypergraph) *A* (Λ-)hypergraph G *is a tuple* (V_G, E_G, c_G, l_G)*, where* V_G *is a set of nodes,* E_G *is a set of edges,* $c_G : E_G \to {V_G}^*$ *is a connection function and* $l_G : E_G \to \Lambda$ *is the labelling function for edges satisfying* $ar(l_G(e)) = |c_G(e)|$ *for every* $e \in E_G$*. Nodes are not labelled.*

A node $v \in V_G$ *is called* isolated *if it is not connected to any edge, i.e., if there are no edges* $e \in E_G$ *and* $u, w \in {V_G}^*$ *such that* $c_G(e) = uvw$*.*

Let G, G' *be (*Λ*-)hypergraphs. A* hypergraph morphism $\varphi : G \to G'$ *consists of a pair of total functions* $\langle \varphi_V : V_G \to V_{G'},\ \varphi_E : E_G \to E_{G'} \rangle$ *such that for every* $e \in E_G$ *it holds that* $l_G(e) = l_{G'}(\varphi_E(e))$ *and* ${\varphi_V}^*(c_G(e)) = c_{G'}(\varphi_E(e))$*. An* edge-bijective *hypergraph morphism is bijective on edges (but not necessarily on nodes). The category of hypergraphs and hypergraph morphisms is denoted by* **Graph**.

In the sequel, we shall often call hypergraphs simply *graphs*, and we will omit the subscripts V and E when referring to the components of a hypergraph morphism.

We introduce graph rewriting rules and their applications to graphs according to the classical Double-Pushout (DPO) approach [22].

Definition 2 (rewriting rule) *A graph rewriting rule is a span of injective graph morphisms* $r = (L \overset{\varphi_L}{\hookleftarrow} K \overset{\varphi_R}{\hookrightarrow} R)$*, where the* left-hand side L*, the* interface K*, and the* right-hand side R *are finite graphs.*

The rewriting rule r is called node-preserving *if (i) φ_L is surjective (and thus bijective) on nodes, (ii) L does not contain isolated nodes, (iii) each isolated node in R belongs to $\varphi_R(K)$. Rule r is* consuming *if (iv) $L - \varphi_L(K)$ is not empty.*

In the paper a rule $r = (L \overset{\varphi_L}{\hookleftarrow} K \overset{\varphi_R}{\hookrightarrow} R)$ will be written simply as $r = (L \hookleftarrow K \hookrightarrow R)$, assuming, without loss of generality, that φ_R and φ_L are inclusions and that $K = L \cap R$. In this case the union $L \cup R$ is well-defined. We next introduce the rewriting mechanism adopted in the paper. This will allow also to clarify the meaning of conditions (i)-(iv) in the definition of rewriting rule.

Definition 3 (graph rewriting) *Let $r = (L \hookleftarrow K \hookrightarrow R)$ be a rewriting rule. A* match *of r in a graph G is a morphism $\varphi : L \to G$, injective on edges. In this case, we write $G \Rightarrow_{r,\varphi} H$ (or simply $G \Rightarrow_r H$) if there exists a diagram*

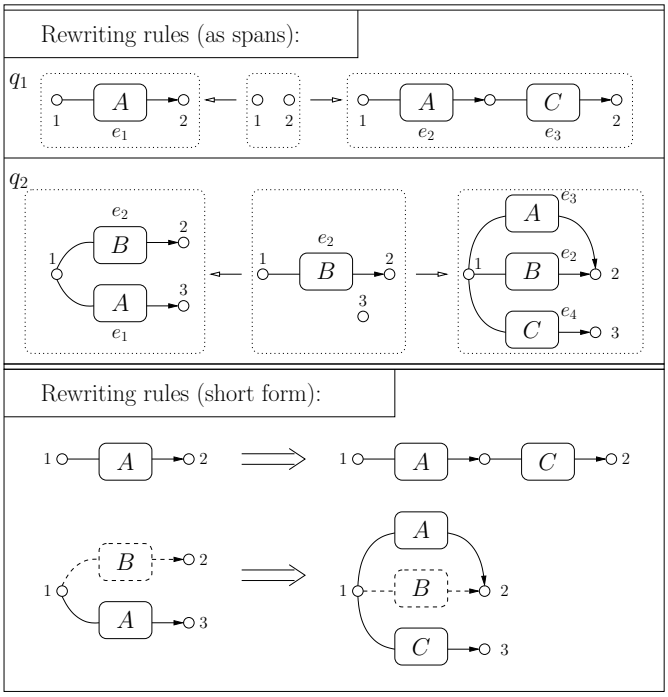
$$\begin{array}{ccccc} L & \hookleftarrow & K & \hookrightarrow & R \\ {\scriptstyle\varphi}\downarrow & & \downarrow & & \downarrow \\ G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

where both squares are pushouts in **Graph**.

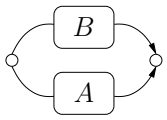
Intuitively, once a match $\varphi(L)$ of a (node-preserving) rule $r = (L \hookleftarrow K \hookrightarrow R)$ is found in a graph G, then G can be rewritten to a graph H that is obtained by first removing the images in G of the edges in $L-K$, and then by adding the items in $R-K$. The images in G of the items in K instead are left unchanged: they are, in a sense, *preserved* or *read* by the rewriting step. Graph D is called the *context* of the rewriting step. For the reader who is familiar with the DPO approach, we remark that there are no application conditions. In fact, the dangling condition and the identification condition are automatically satisfied since rules do not delete nodes and the matches are injective on edges.

Two sample graph rewriting rules are shown in Fig. 1(a). Rule q_1 replaces an edge labelled A with two edges labelled A and C, respectively. The second rule q_2 replaces an edge labelled A again with two edges labelled A and C, but connected in a different way and only if there exists an edge labelled B in the context. For the sake of readability, graphs are enclosed in dotted boxes. If no ambiguity can arise, we usually give rules in their short form, as depicted at the bottom of Fig. 1(a), where the nodes in the interface are numbered, edges in the interface are drawn with dashed lines and edge names disappear. In this running example we consider only binary edges, i.e., edges of arity 2.

Hereafter we shall consider only rules which are node-preserving and consuming. In particular, Condition (i) of Definition 2 guarantees that nodes are never deleted. This is a mild restriction, because the deletion of a node can usually be simulated by leaving the node isolated. Indeed, Conditions (ii) and (iii) essentially state that we are interested only in rewriting up to isolated nodes.



(a) The set of rewriting rules $\mathcal{R} = \{q_1, q_2\}$.



(b) The start graph G_s.

Fig. 1. The running example graph grammar $\mathcal{G}$.

More precisely, by (iii) no node is isolated when created, and by (ii) nodes that become isolated have no influence on further reductions: hence one can safely assume that they are removed by some kind of garbage collection. Finally, Condition (iv) is standard in unfolding-based approaches: every rule must delete some graph items. This ensures that in the unfolding each rule can only be fired once (a fact that will be used later in the technical development).

Definition 4 (graph transformation systems) *A* graph transformation system (GTS) $\mathcal{R}$ *is a finite set of graph rewriting rules. We write* $G \Rightarrow_{\mathcal{R}} H$ *if* $G \Rightarrow_r H$ *for some* $r \in \mathcal{R}$. *Furthermore* $\Rightarrow_{\mathcal{R}}^*$ *denotes the reflexive and transitive closure of* $\Rightarrow_{\mathcal{R}}$. *A* graph grammar *is a pair* $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$, *where* $\mathcal{R}$ *is a* GTS *and* $G_{\mathcal{R}}$ *is (finite) graph, without isolated nodes, called* start graph.

For instance, the rewriting rule q_2 of grammar $\mathcal{G}$ in Fig. 1(a) can be applied to the graph on the left-hand side of Fig. 2, producing the graph on the right-hand side. Both graphs in Fig. 2 are reachable in $\mathcal{G}$ from its start graph G_s depicted in Fig. 1(b). More generally, the graphs reachable in $\mathcal{G}$ consist of several parallel paths: one consisting only of a B-edge, one starting with an A-edge followed by arbitrarily many (possibly 0) C-edges, and arbitrarily many paths consisting only of C-edges. More meaningful examples, modelling distributed systems with process mobility, can be found in Section 5.2.

To simplify later the presentation of Petri graphs it will be useful to have a total ordering on the edges of the graphs in any GTS considered. For this

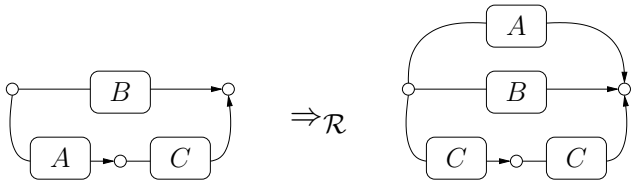


Fig. 2. A graph rewriting step.

reason, we fix throughout the paper a totally ordered set $(\mathbb{E}, \leq)$ and we assume that for any GTS all the involved graphs have edges taken from this set. The ordering can be chosen arbitrarily and is needed in order to distinguish two edges with the same label in a left-hand side. Graph morphisms need not preserve the order, but we will later require that it is preserved by Petri graph morphisms (see Definition 15).

Definition 5 (ordered GTS and grammar) *An* ordered GTS $\mathcal{R}$ *is a* GTS *such that the left-hand side, right-hand side and interface graph of any rewriting rule have edges taken from* $\mathbb{E}$*. An* ordered *graph grammar* $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$ *is a graph grammar such that* $\mathcal{R}$ *is an ordered* GTS *and the edges of the start graph* $G_{\mathcal{R}}$ *are taken from* $\mathbb{E}$*.*

Given a graph G *of an ordered* GTS *(i.e., the start graph or the constituent of a rule) and a subset of its edges* $X = \{e_1, \ldots, e_n\} \subseteq E_G$*, we denote by* $\lambda(X)$ *the* sequence *consisting of the edges in* X *taken according to the total ordering, i.e.,* $\lambda(X) = e_{i_1} \ldots e_{i_n}$*, where* $i_j \in \{1, \ldots, n\}$ *and* $j < h$ *implies* $e_{i_j} < e_{i_h}$*.*

All graph grammars and GTSs in the paper will be implicitly ordered. We remark that this will be useful for presentation issues, but it is inessential for the operational behaviour of graph grammars: matches, rewriting steps and derivations are defined as usual, independently from the ordering of edges. For instance, the graph grammar $\mathcal{G}$ in Fig. 1 can be ordered by assuming that for all edges e_i and e_j, $e_i < e_j$ iff $i < j$.

2.2 Contextual Petri nets and pre-nets

We now fix some basic notation for Petri nets [46,42] and contextual nets [44,58,30], i.e., Petri nets extended with *read arcs*. Then we will briefly discuss pre-nets, a variation of Petri nets introduced in [13].

Given a set A we will denote by $A^{\oplus}$ the free commutative monoid over A, whose elements will be called *multisets* over A. Given a function $f : A \to B$, by $f^{\oplus} : A^{\oplus} \to B^{\oplus}$ we denote its monoidal extension.

On multisets $m, m' \in A^{\oplus}$, we use some common relations and operations, like *inclusion*, defined by $m \leq m'$ when there exists $m'' \in A^{\oplus}$ such that

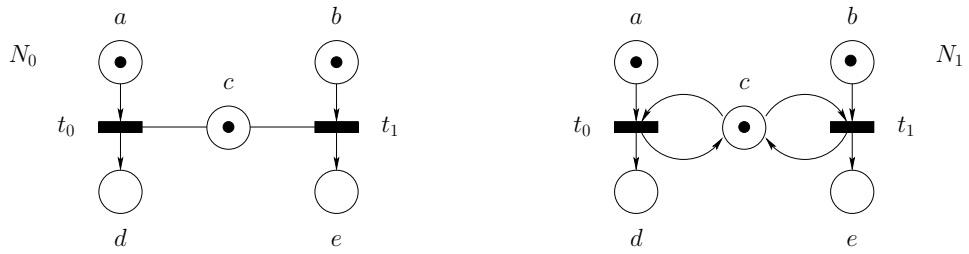


Fig. 3. Ordinary nets do not allow for concurrent read-only operations.

$m \oplus m'' = m'$ and *difference*, which, in the same situation, is defined by $m' - m = m''$. A multiset $m \in A^{\oplus}$ will be sometimes written as a formal sum $m = \bigoplus_{a \in A} m_a \cdot a$ and given m we will write $m(a)$ to denote the coefficient m_a (i.e., $m(a) = \max\{k \mid k \cdot a \leq m\}$). The *join* of two multisets $m \sqcup m'$ is defined as the smallest multiset including m and m', i.e., $\bigoplus_{a \in A} \max\{m_a, m'_a\} \cdot a$. Furthermore, for $m \in A^{\oplus}$ and $a \in A$ we write $a \in m$ for $a \leq m$. The set underlying a multiset $m \in A^{\oplus}$ is defined by $[\![m]\!] = \{a \in A \mid a \in m\}$. Often we will confuse a subset $X \subseteq A$ with the multiset $\bigoplus_{x \in X} x$.

We denote by $\mathbf{m} : A^* \to A^{\oplus}$ the function mapping any sequence to the corresponding multiset. We will write $a \in s$ if a appears in the sequence s, i.e., if $a \in [\![\mathbf{m}(s)]\!]$. Similarly, we write $s_1 \cap s_2$ for $[\![\mathbf{m}(s_1)]\!] \cap [\![\mathbf{m}(s_2)]\!]$.

Let us introduce now contextual Petri nets and their token game.

Definition 6 (contextual Petri net) *Let A be a finite set of action labels. An A-labelled* (contextual) Petri net *is a tuple $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ where S is a set of places, T is a set of transitions, ${}^\bullet(\ ), (\ )^\bullet, \underline{(\ )} : T \to S^{\oplus}$ assign to each transition its pre-set, post-set and context, and $p : T \to A$ assigns an action label to each transition. A* marking *m is a multiset $m \in S^{\oplus}$. A* marked Petri net *is a pair (N, m_N), where N is a Petri net and $m_N \in S^{\oplus}$ is the* initial marking.

Contextual nets are depicted like net N_0 in Fig. 3: circles and boxes represent places and transitions, respectively, directed edges link transitions to places in their pre- and post-sets, while undirected edges represent the read-arcs, connecting the transitions with the places in their contexts. The initial marking m is represented by inserting in any place s the corresponding number $m(s)$ of tokens, depicted as black circles.

Definition 7 (token game) *Given a contextual net $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$, a transition $t \in T$ is* enabled *at a marking $m \in S^{\oplus}$ if ${}^\bullet t \oplus \underline{t} \leq m$. When enabled, the* firing *of t produces a new marking m' obtained by removing the pre-set of t and adding its post-set, i.e., $m' = m - {}^\bullet t \oplus t^\bullet$: in this case we write $m\,[t\rangle\, m'$.*

A firing sequence *of a marked contextual net (N, m_N) is a sequence of fir-*

ings $m_N\,[t_0\rangle\, m_1\,[t_1\rangle\, \cdots m_{n-1}\,[t_{n-1}\rangle\, m_n$ *of transitions of* N*, starting from the initial marking. A marking* m *is* reachable *in* (N, m_N) *if there is a firing sequence ending with* m*; it is* coverable *if there is a firing sequence ending with a marking* m' *such that* $m \leq m'$*.*

A multiset of transitions $U \in T^{\oplus}$ *is concurrently enabled by a marking* $m \in S^{\oplus}$ *if*

$$\bigoplus_{t \in U} U(t) \cdot {}^{\bullet}t \oplus \bigsqcup_{t \in U} \underline{t} \leq m.$$

In this case, the firing of U *produces the new marking*

$$m' = m - \bigoplus_{t \in U} U(t) \cdot {}^{\bullet}t \oplus \bigoplus_{t \in U} U(t) \cdot t^{\bullet}.$$

This is denoted $m\,[U\rangle\, m'$*, and it is called a* step*.*

Intuitively, read arcs allow a transition to check for the presence of a token in a place, without removing the token itself. Furthermore, as just formalised, the same token can be read by several transitions at the same time: in fact, a multiset of transitions $U \in T^{\oplus}$ is concurrently enabled by a marking $m \in S^{\oplus}$ if m contains the sum of all the pre-sets of the transitions in U (each one with its multiplicity) and, additionally, the join of all the contexts of the transitions in U.

Because of this notion of concurrent enabling, the (standard) net obtained from a contextual net by replacing read arcs with self-loops would not be equivalent to the original one: both nets would have the same reachable markings, but the contextual one would allow a greater amount of concurrency. For instance, consider the net N_1 in Fig. 3 and compare it to the net N_0 in the same figure, where place c is connected to transitions t_0 and t_1 by read arcs, meaning that c represents a resource accessed in a read-only manner. While in N_0 the transitions t_0 and t_1 can fire concurrently, in N_1 the two transitions have to be interleaved. In practice the possibility of having concurrent read-only accesses to shared resources can lead to smaller unfoldings and hence to smaller approximations.

For technical reasons, it is convenient in the following to stick to a slightly more concrete model of nets, the so-called *pre-nets*, where a total ordering is imposed on the places occurring in the pre-, post-set and context of transitions. Any pre-net can be seen as a concrete "implementation" of its *underlying* Petri net, obtained by forgetting about the ordering of places. [2]

[2] Pre-nets have been introduced in [13] to obtain a fully satisfactory categorical semantics for nets, where the construction of the model of computation yields an adjunction between the category of nets and the category of models (symmetric monoidal categories).

Total orderings on places will allow us to have a canonical one-to-one correspondence between the places of two transitions related by a morphism (needed for the folding steps introduced later) and to uniquely reconstruct matches (see Proposition 13).

Definition 8 (contextual pre-nets) *Let A be a finite set of action labels. An A-*labelled (contextual) pre-net *is a tuple $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ where S is a set of places, T is a set of transitions, ${}^\bullet(\ ), (\ )^\bullet, \underline{(\ )} : T \to \overline{S^*}$ assign to each transition its pre-set, post-set and context, which are sequences of places, and $p : T \to A$ assigns an action label to each transition.*

A marked Petri pre-net *is a pair (N, u_N), where N is a Petri pre-net and $u_N \in S^*$.*

Observe that, given a transition t in a pre-net, ${}^\bullet t$ and $t^\bullet$ are deliberately called, as for ordinary nets, the pre-*set* and post-*set* of t, although they are not (multi-)sets, but sequences. As in the case of *ordered* graph grammars, the ordering over places is inessential as far as the firing behaviour is concerned. In other words, the token game of a pre-net is defined by referring to the underlying Petri net. For example, we will speak of a *marking* of a pre-net as a multi-set of places, and say that a marking is *reachable* or *coverable* in a pre-net whenever it is reachable or coverable in the underlying Petri net. Also the dependency relations between transitions are defined exactly as in the underlying Petri net, as follows.

We will now define a relation of causal dependence on places and transitions. It will be essential for computing coverings or over-approximations (see Definition 26).

Definition 9 (causality relation) *Let N be a (marked) pre-net. The* causality relation $<_N$ *over N is the least transitive relation on $S \cup T$ such that, for all $t, t' \in T$, $s \in S$, we have (i) $s <_N t$ if $s \in {}^\bullet t$, (ii) $t <_N s$ if $s \in t^\bullet$ and (iii) $t <_N t'$ if $t^\bullet \cap \underline{t'} \neq \emptyset$. For any $x \in S \cup T$ we define its sets of* causes $\lfloor x \rfloor = \{y \in S \cup T \mid y <_N x\}$ *and* consequences $\lceil x \rceil = \{y \in S \cup T \mid x <_N y\}$. *The definitions are extended in the obvious way to subsets X of $S \cup T$, e.g., $\lfloor X \rfloor = \bigcup_{x \in X} \lfloor x \rfloor$.*

A pre-net N is called acyclic *if the relation $<_N$ is acyclic.*

For instance, consider the Petri net N depicted in Fig. 4. It holds that $s_2 <_N t_3 <_N s_3$, furthermore $t_1 <_N s_1 <_N t_2$ and $t_1 <_N t_3$, while s_1 and t_3 are not causally related.

A Petri net satisfies the irredundancy condition if no two distinct transitions have the same label, pre-set and context.

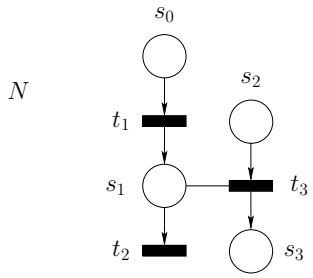


Fig. 4. Causality for contextual nets.

Definition 10 (irredundancy) *A pre-net* $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ *is called* irredundant *if for any* $t, t' \in T$

$$p(t) = p(t') \ \wedge \ {}^\bullet t = {}^\bullet t' \ \wedge \ \underline{t} = \underline{t'} \quad \Rightarrow \quad t = t'. \tag{1}$$

The above property is typically considered in the theory of branching processes of nets [24], where it allows one to interpret each transition of a process as an occurrence of firing of a transition in the original net, uniquely determined by its causal history. Here it will play a role when proving the confluence of the algorithm computing the coverings of graph grammars (see Proposition 39).

2.3 Petri graphs

We now introduce the structures, called *Petri graphs*, that will be used to represent approximations of graph transformation systems. They are a slight variation of the notion introduced in [4], and consist of a graph and of a contextual pre-net whose places are the edges of the graph.

Definition 11 (Petri graph) *Let* $\mathcal{R}$ *be a* GTS. *A Petri graph (for* $\mathcal{R}$*) is a tuple* $P = (G, N)$ *where* G *is a graph,* $N = (E_G, T_N, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p_N)$ *is an irredundant* $\mathcal{R}$*-labelled pre-net where the places are the edges of* G*, and for each transition* $t \in T_N$*, with* $p_N(t) = (L \hookleftarrow K \hookrightarrow R)$*, there exists a graph morphism* $\mu(t) : L \cup R \to G$ *such that*

$${}^\bullet t = \mu(t)^*(\lambda(E_L - E_K)) \ \wedge \ \underline{t} = \mu(t)^*(\lambda(E_K)) \wedge \ t^\bullet = \mu(t)^*(\lambda(E_R - E_K)) \tag{2}$$

Condition (2) guarantees that each transition t in the pre-net can be viewed as an "occurrence" of rule $p_N(t) \in \mathcal{R}$. More precisely, let $p_N(t) = (L \hookleftarrow K \hookrightarrow R)$ and let $\mu(t) : L \cup R \to G$ be the morphism associated with the transition. Then $\mu(t)_{|L} : L \to G$ is a match of the rule in G such that the images in G of the sequences of edges in $L - K$ and K, produced as explained in Definition 5, coincide with the pre-set and context of t, respectively. Furthermore $\mu(t)_{|R} : R \to G$ is a match of the right-hand side such that the image of the sequence

of edges in $R - K$, i.e., the edges produced by the application of the rewriting rule to the considered match in G, coincides with the post-set of t.

Note that the total orderings on places and edges are needed to uniquely reconstruct the matches of left-hand and right-hand sides (see Proposition 13). Without using sequences the morphism μ would not necessarily be unique, especially when the graphs contained in a rule have non-trivial automorphisms.

A sample Petri graph $P = (G, N)$ for the grammar $\mathcal{G}$ of the running example is shown in Fig. 5(a). Transitions are represented by small black rectangles, and the connections between transitions and places/edges are drawn as dashed lines in order to distinguish them from the lines connecting edges and nodes. Transitions t_1 and t_2 correspond to the rewriting rules q_1 and q_2, respectively, i.e., $p_N(t_1) = q_1$ and $p_N(t_2) = q_2$. Although not explicitly represented in the picture, the order of pre- and post-set and contexts of transitions are those induced by edge indexes, for instance, ${}^\bullet t_2 = e'_1$, $\underline{t_2} = e'_2$ and $t_2{}^\bullet = e'_1 e'_3$. Note that there are morphisms $\mu(t_1)$, $\mu(t_2)$ as required in Definition 11. For instance the graph in Fig. 5(b) is the union $L \cup R$ of the left- and right-hand sides of rule q_2, and there exists a morphism $\mu(t_2)$ from this graph to the graph underlying P, mapping edges e_1 and e_3 to e'_1, e_2 to e'_2 and e_4 to e'_3.

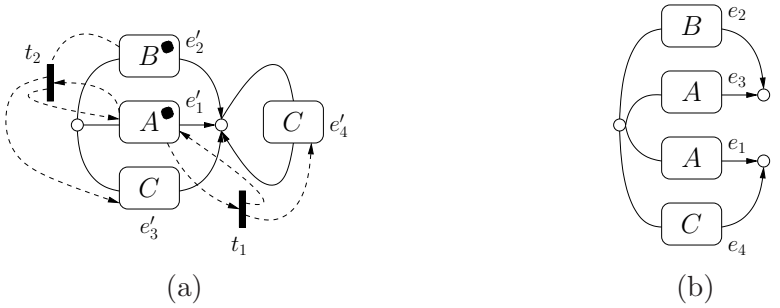


Fig. 5. (a) An example Petri graph $P = (G, N)$ and (b) the graph $L \cup R$ for rule q_2.

A Petri graph for a graph grammar is a Petri graph for the underlying GTS, equipped with an initial state which must correspond to the start graph of the grammar. These Petri graphs will be used to approximate the unfolding of a graph grammar and, as such, they play a role similar to occurrence nets ([24]), where each place represents an occurrence of a token and each transition represents an occurrence of a firing. Therefore, as it happens for occurrence nets, we require that in a Petri graph each item is covered by some reachable marking and each transition can be fired.

Definition 12 (marked Petri graph) *A Petri graph for a graph grammar $\mathcal{G} = (\mathcal{R}, G_\mathcal{R})$, called a* marked Petri graph*, is a pair (P, u) where $P = (G, N)$ is a Petri graph for $\mathcal{R}$ and u is a sequence of places of the Petri graph, called the* initial state*, such that there exists a graph morphism $\iota : G_\mathcal{R} \to G$ with $\iota^*(\lambda(E_{G_\mathcal{R}})) = u$. Furthermore, the following conditions must hold:*

- *every edge of G is* coverable,[3] *and*
- *every transition t of N is* firable, *i.e., there is a coverable marking $m \in {E_G}^{\oplus}$ such that t is enabled at m.*

The Petri graph in our running example in Fig. 5(a) is a Petri graph for the grammar $\mathcal{G}$ in Fig. 1, with an initial state given by the sequence $u = e'_1 e'_2$, indicated by two black tokens.

The notion of Petri graph in this paper is a variation of the one in [4,10]. In the original definition, the underlying net structure was a proper (not contextual) Petri net and the μ components, i.e., the morphisms from rules to the underlying graph, were explicitly given. The next proposition shows that, thanks to the use of pre-nets, in this new setting the μ components are uniquely determined, provided that they exist.

Proposition 13 *Let $\mathcal{R}$ be a* GTS *and let $P = (G, N)$ be a Petri graph for $\mathcal{R}$. Then for any transition $t \in T_N$, with $p_N(t) = (L \hookleftarrow K \hookrightarrow R)$ the graph morphism $\mu(t) : L \cup R \to G$ satisfying Condition (2) of Definition 11 is uniquely determined by ${}^\bullet t$, $\underline{t}$ and $t^\bullet$. Similarly, given a marked Petri graph (P, u) for a graph grammar $(\mathcal{R}, G_\mathcal{R})$ also the graph morphism $\iota : G_\mathcal{R} \to G$ satisfying the condition of Definition 12 is uniquely determined.*

PROOF. (Sketch) The first part of the statement immediately follows from the fact that we work with pre-nets and from the presence of a total ordering on the edges of the graphs of each rewriting rule. The proof also uses the fact that, by conditions (ii) and (iii) of Definition 2, a match of a left-hand side or of the union of left- and right-hand sides of a rule is uniquely determined by the images of the edges.

As for the second statement, it follows from the assumption that there are no isolated nodes in the start graph of any graph grammar. □

Notation. *Given a Petri graph $P = (G, N)$ for a* GTS *$\mathcal{R}$, in the following we will write $\mu(t)$ to denote the unique graph morphism $\mu(t) : L \cup R \to G$ satisfying condition (2) of Definition 11. Similarly, for a marked Petri graph (P, u) for a graph grammar $(\mathcal{R}, G_\mathcal{R})$ we will denote by ι the unique graph morphism $\iota : G_\mathcal{R} \to G$ satisfying the condition in Definition 12.*

The above considerations motivate the use of pre-nets in place of ordinary Petri nets. In fact, without pre-nets the above mentioned morphisms μ (and ι) would not be unique since a left-hand side (the start graph) might contain several

[3] A marking $m \in {E_G}^{\oplus}$ is called *reachable (coverable)* in (P, u), with $P = (G, N)$, if it is reachable (coverable) in the underlying marked pre-net (N, u).

edges with the same label. However, for the completeness of the unfolding (see Proposition 24), it is necessary to distinguish among edges with the same label which occur in the left-hand side of a rewriting rule. Thus, without pre-nets, the functions μ and ι should be explicitly part of the Petri graph, making the presentation heavier.

Given a Petri graph $P = (G, N)$, every marking $m \in {E_G}^{\oplus}$ identifies a graph. A *safe* marking m (i.e., such that $m(e) \leq 1$ for all $e \in E_G$) is intended to represent the subgraph of G consisting of the edges in m and of the nodes attached to these edges. For general markings, edges with multiplicity k will result in k "parallel" edges. This is formalised in the next definition.

Definition 14 (graph generated by a marking) *Let $P = (G, N)$ be a Petri graph and let $m \in {E_G}^{\oplus}$ be a marking of N. The* graph generated by *m, denoted $graph_G(m)$, is the graph H defined as follows: $V_H = \{v \in V_G \mid \exists e \in m \colon v \in c_G(e)\}$, $E_H = \{(e, i) \mid e \in m \wedge 1 \leq i \leq m(e)\}$, $c_H((e, i)) = c_G(e)$ and $l_H((e, i)) = l_G(e)$.*

In other words, $graph_G(m)$ is obtained from G by first removing all edges that are not covered by m, then multiplying all edges according to the number of times they appear in m, and finally removing all isolated nodes. For instance, the marking of the Petri graph in Fig. 5(a) generates the start graph of the running example, depicted in Fig. 1(b).

Observe that $graph_G(m)$ is the only graph, up to isomorphism, which has no isolated nodes and for which there exists a graph morphism $\varphi \colon graph_G(m) \to G$ injective on nodes such that $\varphi^{\oplus}(E_{graph_G(m)}) = m$.

In the following we will sometimes confuse a marking of a Petri graph with its generated graph, saying for example that a given graph is reachable in a Petri graph.

For the technical development of the paper, it is convenient to look at (marked) Petri graphs as objects of suitable categories, that we are going to define by introducing a notion of Petri graph morphisms. This will allow us to characterise the results of certain operations on Petri graphs as colimits of suitable diagrams, and later to formalise in which sense a chain of approximations will have the full unfolding of a graph grammar as its limit.

Definition 15 (categories of Petri graphs) *Let $P = (G, N)$, $P' = (G', N')$ be Petri graphs for a given* GTS *$\mathcal{R}$. A* Petri graph morphism *is a pair $\psi = (\psi_G, \psi_N) : P \to P'$ where*

- *$\psi_G : G \to G'$ is a graph morphism;*

- $(\psi_G|_{E_G}, \psi_N) : N \to N'$ *is a* labelled pre-net morphism*, that is,* $\psi_N : T_N \to T_{N'}$ *is a mapping such that for every* $t \in T_N$, ${}^\bullet\psi_N(t) = \psi_G{}^*({}^\bullet t)$, $\psi_N(t)^\bullet = \psi_G{}^*(t^\bullet)$, $\underline{\psi_N(t)} = \psi_G{}^*(\underline{t})$*, and* $p_{N'} \circ \psi_N = p_N$.

The category of Petri graphs for $\mathcal{R}$ *and Petri graph morphisms is denoted by* $\mathsf{PG}(\mathcal{R})$.

The category of marked Petri graphs for a graph grammar $\mathcal{G}$ *and morphisms* $\psi\colon (P, u) \to (P', u')$ *which preserve initial states, i.e., such that* $\psi_G{}^*(u) = u'$*, is denoted by* $\mathsf{PG}_\iota(\mathcal{G})$.

In the following we will often omit the subscripts G and N. Moreover when the GTS $\mathcal{R}$ or the graph grammar $\mathcal{G}$ are clear from the context, the corresponding Petri graph categories will be denoted simply by PG and PG_ι.

As an example, Fig. 6 shows a second Petri graph P' for our running example grammar $\mathcal{G}$ in Fig. 1. This can be mapped to the Petri graph in Fig. 5(a) via a Petri graph morphism which maps transitions t_1 and t_2 of the source Petri graph to the corresponding transitions in the target. Concerning edges, the morphism maps e_1'', e_5'' and e_6'' to e_1', e_2'' to e_2', e_3'' to e_3', and e_4'' to e_4'.

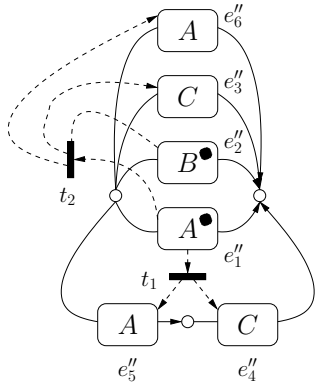


Fig. 6. Another sample Petri graph.

We shall often exploit the following important property of categories $\mathsf{PG}(\mathcal{R})$ and $\mathsf{PG}_\iota(\mathcal{G})$.

Proposition 16 (cocompleteness of Petri graph categories) *Let* $\mathcal{R}$ *be a* GTS *and* $\mathcal{G}$ *be a graph grammar. Then the category of Petri graphs* $\mathsf{PG}(\mathcal{R})$ *and the category of marked Petri graphs* $\mathsf{PG}_\iota(\mathcal{G})$ *are both cocomplete, i.e., they have all colimits.*

PROOF. See the Appendix. □

Roughly, in order to construct the colimit first the pointwise colimit is taken on nodes, edges and transitions, and then the resulting structure is quotiented in order to fulfil the irredundancy condition of Petri graphs. In particular we will later make use of pushouts and coequalizers to define unfolding and folding operations.

3 Unfolding and under-approximations

In this section we define the unfolding of a graph grammar. Following a common approach in the literature (see, e.g., [51,54]) the unfolding is defined as the union (categorically, the colimit) of the chain of its finite prefixes, each of which can be seen as an *under-approximation* of the behaviour of the system.

The finite prefixes of the unfolding are constructed incrementally beginning from the start graph and then applying at each step in all possible ways the rules, without deleting the left-hand sides, and recording each occurrence of a rule and each new graph item generated in the rewriting process. The process stops at a given causal depth.

To define a basic unfolding step, we first need to fix some notation. Every graph G can be considered as a Petri graph $[G] = (G, N)$ for any GTS $\mathcal{R}$, by taking N as the net with places E_G and no transitions. Similarly, G can be seen as a marked Petri graph $((G, N), u)$ for the graph grammar $(\mathcal{R}, G)$, by taking N as above and $u = \lambda(E_G)$ as the initial state. If $P = (N, G)$ is a Petri graph and $\varphi\colon G' \to G$ is a graph morphism then we will use the same symbol $\varphi\colon [G'] \to P$ to denote the corresponding Petri graph morphism.

Moreover, if $r = (L \hookleftarrow K \hookrightarrow R)$ is a rule, we will write $P(r)$ to denote the Petri graph $(L \cup R, N)$ where $N = (E_{L\cup R}, \{t\}, {}^\bullet t = \lambda(E_L - E_K), t^\bullet = \lambda(E_R - E_K), \underline{t} = \lambda(E_K), p_N(t) = r)$. For instance the Petri graph $P(r)$ for rule q_2 in Fig. 1(a) is depicted in Fig. 7. Intuitively it provides an alternative representation of a rule where consumption, preservation and deletion of edges is represented in the Petri net notation.

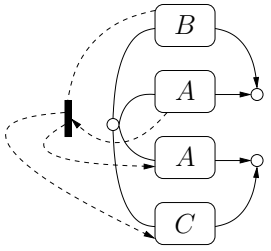


Fig. 7. A Petri graph for rewriting rule q_2 of Fig. 1(a).

Definition 17 (unfolding operation) *Let $P = (G, N)$ be a Petri graph for*

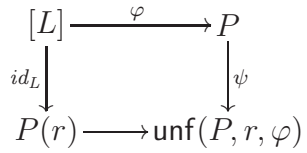


Fig. 8. Diagram for an unfolding step.

a GTS $\mathcal{R}$. *Let* $r = (L \hookleftarrow K \hookrightarrow R) \in \mathcal{R}$ *be a rule and let* $\varphi : L \to G$ *be a match of* r *in* G. *The* unfolding *of* P *with rule* r *at match* φ, *denoted* $\mathsf{unf}(P, r, \varphi)$, *is the Petri graph obtained as pushout of* $\varphi : [L] \to P$ *and* $id_L : [L] \to P(r)$ *(see Fig. 8).*

If (P, u) *is a marked Petri graph for a graph grammar* $(\mathcal{R}, G_{\mathcal{R}})$ *and* $\varphi^{\oplus}(E_L)$ *is coverable, in the same situation, we define* $\mathsf{unf}((P, u), r, \varphi) = (P', \psi^*(u))$, *where* $P' = \mathsf{unf}(P, r, \varphi)$ *and* $\psi : P \to P'$ *is the* PG *morphism in the pushout diagram defining the unfolding operation (see Fig. 8).*

Roughly, given a Petri graph $P = (G, N)$, whenever a rule r admits a match φ in G, the unfolding operation allows to extend P by adding an occurrence of rule r at match φ. The resulting Petri graph $\mathsf{unf}(P, r, \varphi)$ is obtained by merging P with the Petri graph $P(r)$, representing the rule, along the match. Categorically, this corresponds to a pushout. Fig. 9 shows two unfolding steps for the running example grammar $\mathcal{G}$. Matches are specified by depicting the involved edges in grey.

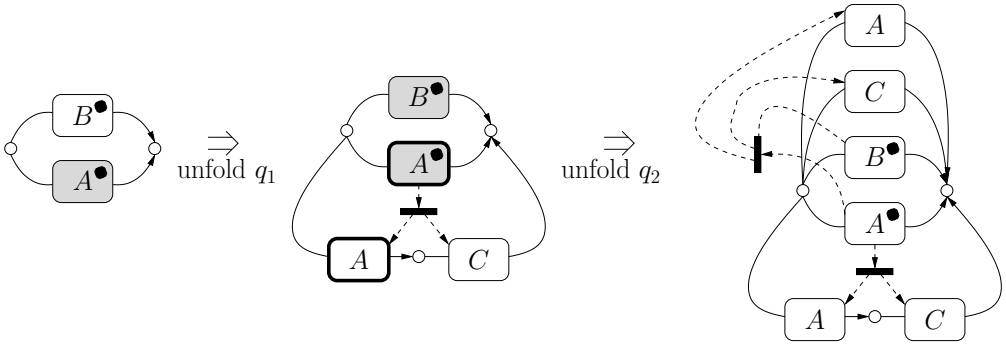


Fig. 9. Two unfolding steps for the running example grammar $\mathcal{G}$ in Fig. 1

Observe that in the marked case, since $\varphi^{\oplus}(E_L)$ is coverable, using the fact that, by Definition 12, in (P, u) all places can be covered and all transitions can be fired, we can prove that the same holds in the resulting Petri graph $\mathsf{unf}((P, u), r, \varphi)$. Hence $\mathsf{unf}((P, u), r, \varphi)$ is a well-defined marked Petri graph.

We introduce now the *depth* of an item in a Petri graph. To deal with the presence of causal cycles we take as codomain of the depth function the complete partial order [4] of natural numbers extended with "infinity".

[4] Recall that a partial order is called complete if each directed subset has a least

Definition 18 *We denote by* $\mathbb{N}^\omega$ *the partially ordered set* $(\mathbb{N} \uplus \{\omega\}, \leq)$ *where* $\leq$ *is the usual order on natural numbers and* $n \leq \omega$ *for any* $n \in \mathbb{N}$*. Addition is extended to* $\mathbb{N}^\omega$ *by continuity, i.e., for* $m, n \in \mathbb{N}^\omega$*:*

$$m + n = \begin{cases} m + n & \text{if } m, n \in \mathbb{N} \\ \omega & \text{otherwise} \end{cases}$$

We start with a definition of depth over Petri nets, which is later extended to Petri graphs. The function *depth* assigns to each item x of a pre-net its causal depth, i.e., the length of the maximal chain of causally related items leading from items without causes to x. In particular, an item x located in a causality cycle will have an infinite depth, i.e., $\mathit{depth}(x) = \omega$.

Definition 19 (depth of items of a Petri net) *Let* N *be a pre-net. Let* D_N *be the function* $D_N : (S_N \cup T_N \to \mathbb{N}^\omega) \to (S_N \cup T_N \to \mathbb{N}^\omega)$ *defined as follows, where* $\bigsqcup X$*, for* $X \subseteq N^\omega$*, denotes the least upper bound of the set* X*:*

$$D_N(d)(x) = \begin{cases} \bigsqcup\{d(t) \mid t \in T_N \ \wedge \ x \in t^\bullet\} & \text{if } x \in S_N \\ \bigsqcup\{d(s) \mid s \in S_N \ \wedge \ s \in {}^\bullet x \cdot \underline{x}\} + 1 & \text{if } x \in T_N \end{cases}$$

Then the function $\mathit{depth} : S_N \cup T_N \to \mathbb{N}^\omega$ *that assigns depth information to every item of* N *is defined as* $\mathit{depth} = \mathit{fix}(D_N)$*, i.e., the least fixed point of* D_N*.*

Observe that $S_N \cup T_N \to \mathbb{N}^\omega$, endowed with the pointwise order, is a complete partial order, since $\mathbb{N}^\omega$ is itself complete. Moreover D_N is monotone and continuous, and thus *depth* is the least upper bound of the chain $\langle D_N^n(\mathbf{0})\rangle_{n\in\mathbb{N}}$ obtained by iterating D_N over the constant function $\mathbf{0}$ mapping every item to 0. This follows directly from the fixpoint theorem for complete partial orders [18].

If the considered pre-net is finite, then the depth function can be computed constructively. In fact, it is easy to check that the least fixed point can be obtained by iterating the operator D_N defined above on the zero function $n = 2 \cdot (|T_N| + 1)$ times. For any item x, if $D_N^n(\mathbf{0})(x) = h \leq |T_N|$ then $\mathit{depth}(x) = h$ else $\mathit{depth}(x) = \omega$.

The definition is extended to Petri graphs in a straightforward way: Places become edges and we have to take into account also the presence of nodes.

Definition 20 (depth of items in a Petri graph) *Let* $P = (G, N)$ *be a Petri graph. The function* $\mathit{depth} : E_G \cup T_N \to \mathbb{N}^\omega$ *is defined as in Definition 19.*

upper bound.

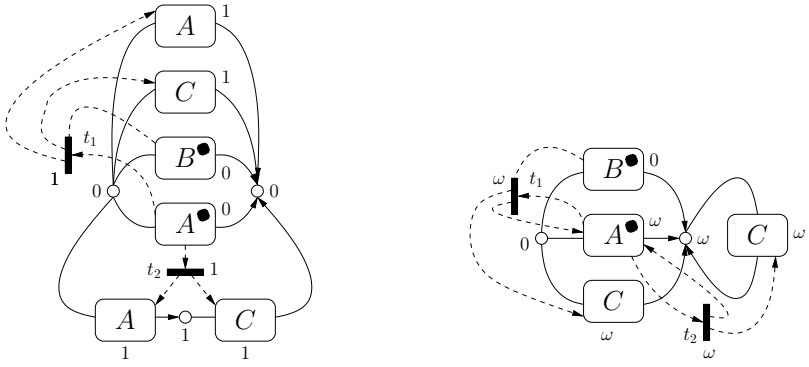
This function is extended to nodes by defining, for $v \in V_G$

$$depth(v) = \bigsqcup\{depth(t) \mid t \in T_N \wedge v \in \mu(t)(V_R - V_K)\}$$

where for each $t \in T_N$ *with* $p_N(t) = (L \hookleftarrow K \hookrightarrow R)$, $\mu(t) : L \cup R \to G$ *is the unique morphism which exists by definition of Petri graph (see Definition 11 and Proposition 13).*

Therefore the depth of a node v is the maximal depth of rules with left-hand side L and right-hand side R where v appears in $R - L$, i.e., intuitively, of rules which can "generate" node v.

As an example, Fig. 10 shows two Petri graphs seen before, enriched with the indication of the depth of the various items.



(a) (b)

Fig. 10. Depth of items in a Petri graph.

We are now ready to define the prefix of the unfolding of a graph grammar up to a given causal depth k, called the k-truncation of the unfolding.

Definition 21 (algorithm for k-truncation) *Let* $k \in \mathbb{N}$ *and let* $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$ *be a graph grammar. The algorithm generates a sequence* $(P_i, u_i)_{i \in \mathbb{N}}$ *of Petri graphs, as follows.*

(Step 0) *Initialise* $(P_0, u_0) = ([G_{\mathcal{R}}], \lambda(E_{G_{\mathcal{R}}}))$.

(Step $i+1$) *Let* (P_i, u_i), *with* $P_i = (G_i, N_i)$, *be the Petri graph produced at step* i.

⋆ *Unfolding: Find a rule* $r = (L \hookleftarrow K \hookrightarrow R)$ *in* $\mathcal{R}$ *and a match* $\varphi : L \to G_i$ *such that*
- $\varphi^{\oplus}(E_L)$ *is a coverable marking in* P_i*;*
- *there is no transition* $t \in T_{N_i}$ *such that* $p_{N_i}(t) = r$ *and* ${}^\bullet t = \varphi^*(\lambda(E_L - E_K))$ *and* $\underline{t} = \varphi^*(\lambda(E_K))$*;*

· *for any edge or node* $x \in \varphi(L)$ *it holds that* $depth(x) < k$.
Then set $(P_{i+1}, u_{i+1}) = \mathsf{unf}((P_i, u_i), r, \varphi)$.

If no unfolding step can be performed, the algorithm terminates. The resulting marked Petri graph is called k-truncation *of the unfolding of* $\mathcal{G}$ *and denoted by* $\mathcal{T}^k(\mathcal{G})$.

For any i *we will denote by* $\psi_i : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ *the* PG_ι *morphism arising in the unfolding operation (see Definition 17).*

It can be easily proved that the unfolding procedure described above is terminating and confluent (up to isomorphisms). The proof of termination essentially relies on the finiteness of start graph, set of rules and right-hand sides of rules, and on the fact that the grammar is consuming. Confluence is an easy consequence of the commutativity of colimits. As an example, the 1-truncation of the running example grammar in Fig. 1 is the Petri graph in Fig. 10(a). Note that it can be obtained from the start graph executing two unfolding steps, corresponding to the application of the two rules of the grammar.

For any $k \in \mathbb{N}$ we consider a Petri graph morphism $\lambda_k : \mathcal{T}^k(\mathcal{G}) \to \mathcal{T}^{k+1}(\mathcal{G})$, defined as follows. Let $P_0, \ldots, P_n = \mathcal{T}^{k+1}(\mathcal{G})$ be a sequence of Petri graphs generated by the algorithm of Definition 21 for the construction of $\mathcal{T}^{k+1}(\mathcal{G})$. By confluence of the unfolding procedure, we can assume that all the unfolding steps up to level k are performed first, and thus that there exists $j \leq n$ such that $P_j = \mathcal{T}^k(\mathcal{G})$. In this case we define $\lambda_k = \psi_{n-1} \circ \ldots \circ \psi_j$, where ψ_i (for $i \in \{j, \ldots, n-1\}$) is as in Definition 21.

Definition 22 (truncation tower) *The following diagram, where the* λ_i *are defined as above,*

$$\mathcal{T}^0(\mathcal{G}) \stackrel{\lambda_0}{\to} \ldots \mathcal{T}^k(\mathcal{G}) \stackrel{\lambda_k}{\to} \mathcal{T}^{k+1}(\mathcal{G}) \stackrel{\lambda_{k+1}}{\to} \ldots$$

is called the truncation tower *of graph grammar* $\mathcal{G}$.

The next definition introduces the full unfolding of a graph grammar as the colimit of its finite truncations, which exists by cocompleteness.

Definition 23 (unfolding as colimit of the k-truncations) *The* (full) unfolding $\mathcal{U}(\mathcal{G})$ *of a graph grammar* $\mathcal{G}$ *is the colimit in the category* PG_ι *of its truncation tower.*

As mentioned in the introduction, the unfolding fully represents the concurrent behaviour of the original grammar: the possible applications of rewriting rules which can appear in any computation, the causal dependencies and the conflicts (mutual exclusion) between such applications. This is formalised by expressing the unfolding construction as a categorical adjunction (see [12,9]).

The unfolding that we obtain here is exactly the one presented in the cited papers, up to a different syntactical presentation.

In particular, the proposition below states a property of the unfolding which is crucial for this paper: Any graph reachable in a graph grammar can be mapped *isomorphically* to a reachable subgraph of its unfolding, and, vice versa, any reachable subgraph of the unfolding is the isomorphic image of a reachable graph in the original grammar. Furthermore, steps in the original grammar correspond to steps in the unfolding.

Proposition 24 (reachable graphs are subgraphs of the unfolding) *Let $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar and let $\mathcal{U}(\mathcal{G}) = ((U, N), u)$ be its unfolding. Let $\mathbb{G}$ be the set of graphs reachable in $\mathcal{G}$, let $\mathbb{M}$ be the set of markings of N reachable from the marking $m_0 = \mathbf{m}(u)$, corresponding to the start graph of $\mathcal{G}$.*

Then there exists a relation $\mathcal{B} \subseteq \mathbb{G} \times \mathbb{M}$, satisfying the following properties:

(i) $(G_{\mathcal{R}}, m_0) \in \mathcal{B}$.

(ii) $\mathcal{B}$ *is a* bisimulation, *i.e., for any* $(G, m) \in \mathcal{B}$,
- *if* $G \Rightarrow_r G'$, *then there are a transition* t *in* $\mathcal{U}(\mathcal{G})$ *and a marking* m' *such that* $m\,[t\rangle\, m'$, $p_N(t) = r$, *and* $(G', m') \in \mathcal{B}$.
- *if* $m\,[t\rangle\, m'$, *then there is a graph* G' *such that* $G \Rightarrow_{p_N(t)} G'$ *and* $(G', m') \in \mathcal{B}$.

(iii) For every pair $(G, m) \in \mathcal{B}$ *there exists an injective morphism* $\varphi_G \colon G \to U$ *such that* $\varphi^{\oplus}(E_G) = m$.

PROOF. The proof is lengthy, but it follows quite straightforwardly from the construction. □

Obviously, for each $k \in \mathbb{N}$ the k-truncation $\mathcal{T}^k(\mathcal{G})$ represents, in general, only an under-approximation of the behaviour of the original grammar $\mathcal{G}$. More precisely, there exists a relation $\mathcal{B}^k$ between the graphs reachable in $\mathcal{G}$ and the markings of $\mathcal{T}^k(\mathcal{G})$ which is a *simulation*: it satisfies conditions *(i)* and *(iii)* of Proposition 24, but only the second part of condition *(ii)*. Actually, the first part of condition *(ii)* also holds, but only for graphs reachable in $\mathcal{G}$ in at most k (possibly concurrent) steps. Still, as we will see in Section 5.2, k-truncations can be useful for proving some interesting properties of the original grammar.

4 Folding operation and over-approximations

In this section we present an algorithm which, given a graph grammar $\mathcal{G}$ and a level of accuracy k, produces a finite Petri graph $\mathcal{C}^k(\mathcal{G})$, called *$k$-covering*, which can be seen as an *over-approximation* of the behaviour of the grammar $\mathcal{G}$.

We have already mentioned that the full unfolding is usually infinite. To obtain a finite over-approximation we modify the unfolding procedure by considering, besides the unfolding operation, also a *folding operation* which allows us to "merge" two occurrences of the left-hand side of a rule whenever one of them causally depends, in a sense made precise later, on the other. Intuitively, the presence of such two occurrences of a left-hand side indicates a cyclic behaviour and applying the folding rule one avoids to unfold the corresponding infinite path. While guaranteeing finiteness, the folding operation causes a loss of information, in the sense that the resulting structure over-approximates the behaviour of the original system: As it happens in the full unfolding, every graph reachable in the original grammar $\mathcal{G}$ corresponds to a marking which is reachable in the covering and every valid derivation in $\mathcal{G}$ corresponds to a valid firing sequence in the covering, but there are reachable markings and valid firing sequences in the covering which have no counterpart in the grammar.

In order to compute better over-approximations of the behaviour, the idea is to delay folding steps, constraining the algorithm to apply only unfolding steps until a given causal depth is reached. Roughly, this is obtained by "freezing" an initial part of the approximated unfolding, up to a given causal depth k, and by allowing only unfolding and no folding steps to affect that part. The resulting over-approximation $\mathcal{C}^k(\mathcal{G})$ is "exact" up to causal depth k, in the sense that any graph reachable in $\mathcal{G}$ in less than k (possibly concurrent) steps will have a reachable *isomorphic* image in $\mathcal{C}^k(\mathcal{G})$. Instead, graphs which are reachable in a larger number of steps, in general, will be mapped only homomorphically in $\mathcal{C}^k(\mathcal{G})$.

In this way one can obtain arbitrarily accurate approximations, a fact which is formalised by proving that the chain of k-coverings for a grammar $\mathcal{G}$ converges to the full unfolding $\mathcal{U}(\mathcal{G})$. In categorical terms, $\mathcal{U}(\mathcal{G})$ is shown to be the limit of the chain of coverings in the category of marked Petri graphs.

4.1 Computing k-coverings

The folding operation is the new ingredient needed to introduce the algorithm for computing k-coverings.

Definition 25 (folding operation) *Let $P = (G, N)$ be a Petri graph for a* GTS $\mathcal{R}$. *Let $r = (L \hookleftarrow K \hookrightarrow R) \in \mathcal{R}$ be a rule and let $\varphi', \varphi : L \to G$ be matches of r in G. The* folding *of P at the matches φ', φ, denoted by $\mathsf{fold}(P, r, \varphi', \varphi) = P'$, is the Petri graph P', equipped with a morphism $\psi : P \to P'$, obtained as the coequalizer of $\varphi, \varphi' : [L] \to P$ in the category* PG.[5]

If (P, u) is a marked Petri graph for a graph grammar $(\mathcal{R}, G_{\mathcal{R}})$, in the same situation, we define $\mathsf{fold}((P, u), r, \varphi', \varphi) = (P', \psi^*(u))$ where P' and $\psi : P \to P'$ are as above.

Roughly speaking, the folding operation merges the two matches of rule r in P producing a new Petri graph P'. Actually, due to the irredundancy requirement for Petri graphs, as a side-effect of the identification of the images of φ and φ', other items can be merged. Note that whenever two transitions are merged, the order of places in the post-set is quite relevant since it determines how the items in the post-sets are to be merged.

We can now describe the algorithm which produces the k-covering $\mathcal{C}^k(\mathcal{G})$ of the unfolding of a graph grammar $\mathcal{G}$. The algorithm generates a sequence of Petri graphs, beginning from the start graph of $\mathcal{G}$ and applying at each step, non-deterministically, a folding or unfolding operation, until no such steps are possible anymore. Folding steps will be applied only at depth k or greater. Note that as soon as folding steps are applied, the Petri graph will contain cycles.

Definition 26 (algorithm for k-covering) *Let $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar and let $k \in \mathbb{N}$. The algorithm generates a sequence $(P_i, u_i)_{i \in \mathbb{N}}$ of Petri graphs, as follows.*

(Step 0) *Initialise $(P_0, u_0) = ([G_{\mathcal{R}}], \lambda(E_{G_{\mathcal{R}}}))$.*

(Step $i+1$) *Let (P_i, u_i), with $P_i = (G_i, N_i)$, be the Petri graph produced at step i. Choose non-deterministically one of the following actions*

- ⋆ ***Folding:*** *Find a rule $r = (L \hookleftarrow K \hookrightarrow R)$ in $\mathcal{R}$ and two matches $\varphi', \varphi : L \to G_i$ of r such that*
  (F1) *φ' and φ are different.*
  (F2) *$\varphi^{\oplus}(E_L)$ is a coverable*[6] *marking in P_i;*

[5] Existence of the coequaliser is ensured by Proposition 16. More explicitly, the pair $\langle P', \psi \rangle$ is characterised uniquely, up to isomorphism, by the fact that $\psi \circ \varphi' = \psi \circ \varphi$, and that for any other pair $\langle P'', \psi'' : P \to P'' \rangle$ such that $\psi'' \circ \varphi' = \psi'' \circ \varphi$ there exists a unique $k : P' \to P''$ such that $k \circ \psi = \psi''$.

[6] Note that coverability may be expensive to check once the Petri graph has been folded. Incremental techniques for coverability checking are described in [35].

(F3) *there exists a transition* $t \in T_{N_i}$ *such that*
(i) $p_{N_i}(t) = r \ \wedge \ \varphi'^*(\lambda(E_L - E_K)) = {}^\bullet t \ \wedge \ \varphi'^*(\lambda(E_K)) = \underline{t}$;
(ii) $\forall e \in E_L : (\varphi(e) = \varphi'(e) \ \vee \ t <_{N_i} \varphi(e))$.
(F4) *for every edge or node* $x \in E_L \cup V_L$ *it holds that*

$$\varphi(x) = \varphi'(x) \ \vee \ (depth(\varphi(x)) \geq k \ \wedge \ depth(\varphi'(x)) \geq k).$$

Then set $(P_{i+1}, u_{i+1}) = \mathsf{fold}((P_i, u_i), r, \varphi', \varphi)$.

⋆ ***Unfolding:*** *Find a rule* $r = (L \hookleftarrow K \hookrightarrow R)$ *in* $\mathcal{R}$ *and a match* $\varphi : L \to G_i$ *such that*
(U1) $\varphi^{\oplus}(E_L)$ *is a coverable marking in* P_i;
(U2) *there is no transition* $t \in T_{N_i}$ *such that* $p_{N_i}(t) = r \ \wedge \ {}^\bullet t = \varphi^*(\lambda(E_L - E_K)) \ \wedge \ \underline{t} = \varphi^*(\lambda(E_K))$;
(U3) *there is no other match* $\varphi' : L \to G_i$ *such that the pair* φ', φ *satisfies the folding condition.*
Then set $(P_{i+1}, u_{i+1}) = \mathsf{unf}((P_i, u_i), r, \varphi)$.

If no folding or unfolding step can be performed, the algorithm terminates. The resulting marked Petri graph is called k-covering *of the unfolding of* $\mathcal{G}$ *and denoted by* $\mathcal{C}^k(\mathcal{G})$.

For any i *we will denote by* $\psi_i : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ *the* PG_ι *morphism arising in the unfolding or folding operations (see Definitions 17 and 25).*

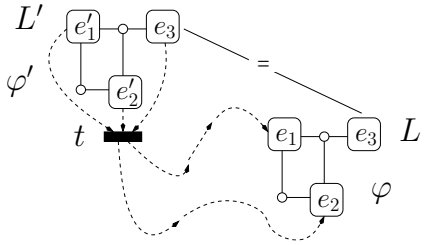


Fig. 11. Graphical representation of the folding condition (F3).

Condition (F3) basically states that we can fold two matches φ', φ of a rule r whenever the first one has been already unfolded producing a transition t, and the second match depends on the first one, in the sense that any edge in the second match appears already in the first one or causally depends on t. A graphical intuition of this condition is given in Fig. 11. We can see two matches L' and L of the same left-hand side, both consisting of three interconnected edges. The folding condition is satisfied since the first match L' has been already unfolded, generating transition t, and the first two edges e_1, e_2 of L are causally dependent on t, while the third one e_3 coincides with the corresponding edge of L'.

It is also possible to drop this condition, obtaining however less precise over-approximations.

Roughly, the idea is that we should not unfold a left-hand side if, in its causal history, we have already done the same unfolding step, since this might lead to infinitely many steps. A similar idea is developed in [28], where the sets of descendants and of normal forms of term rewriting systems are approximated by constructing an approximation automaton.

It is worth stressing that the equality $\varphi(e) = \varphi'(e)$ in the disjunction in condition (F3) is really needed. Its omission would permit to construct an input grammar on which the algorithm does not terminate. For instance, in the example of Fig. 12, which is commented in more detail later, we would never be allowed to fold two matches for rule q_2, since all transitions labelled by this rule contain the same B-edge in their context.

Notice also that by Condition (F4) only items of depth k or greater can be merged. This ensures that the prefix up to depth k of the unfolding is not involved in any folding operation. Actually, in order to guarantee termination, some items of depth smaller than k can be involved in a folding operation, but they will be left unchanged by the step (intuitively they are merged with themselves).

A main result of this paper, presented next in Section 4.2, is that the non-deterministic algorithm just described is terminating and confluent, and thus for any $k \geq 0$ the k-covering of a graph grammar is uniquely determined, up to isomorphisms.

Fig. 12 shows two possible runs of the non-deterministic algorithm of Definition 26 applied to the running example grammar $\mathcal{G}$, with $k = 0$. As a consequence of the confluence result proved in the next section, both runs terminate with the same Petri graph. It is worth stressing that the irredundancy condition for Petri graphs is essential for this result. A step where the effect of this condition can be seen is marked in Fig. 12 by "irredundancy!": in this step, because of irredundancy, not only the two bottom A-edges are merged, but also the two transitions consuming them and hence the two top A-edges and two C-edges.

4.2 Termination, confluence and over-approximation

In this section, after some technical preliminaries, we shall first prove that the algorithm described in Definition 26 is terminating and confluent. Hence, by a classical result, the k-covering of a graph grammar is uniquely determined, up to isomorphisms. Next we show that the algorithm produces over-approximations of the original graph grammar, namely, that for a given graph grammar $\mathcal{G}$, if we consider the k-covering $\mathcal{C}^k(\mathcal{G})$ produced by the algorithm

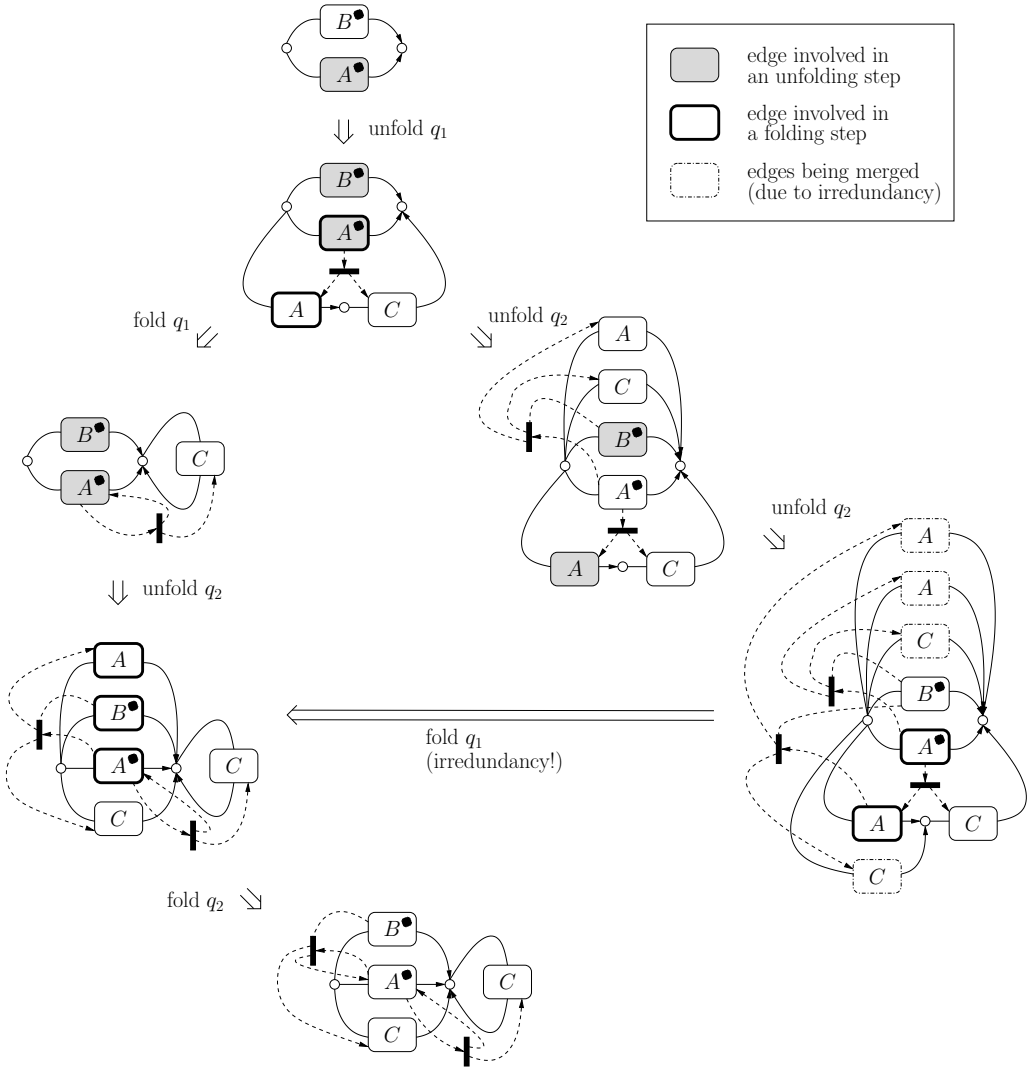


Fig. 12. Two different computations leading to the same 0-covering.

then every graph reachable in $\mathcal{G}$ has a homomorphic image in $\mathcal{C}^k(\mathcal{G})$. Finally, we conclude by showing that $\mathcal{C}^k(\mathcal{G})$ is "exact" up to causal depth k.

Let us start by introducing a few technical definitions and lemmata that will be exploited in the main proofs.

Lemma 27 *Let $\psi : P \to P'$ be a* PG *morphism where $P = (G, N)$. Then for any node, edge or transition $x \in V_G \cup E_G \cup T_N$ it holds that $depth(\psi(x)) \geq depth(x)$.*

PROOF. See the Appendix. □

We introduce now the notions of *k-monomorphisms* and *k-isomorphisms* between Petri graphs. These are morphisms that are injective (respectively, bi-

jective) when restricted to items of depth smaller than k in the source and target Petri graphs, and that preserve the depth of such items. Next we will show that the morphisms ψ_i relating the marked Petri graphs (P_i, u_i) generated at successive steps of the construction of the k-covering are indeed k-monomorphisms.

Definition 28 (k-monomorphism, k-isomorphism) *Let $\psi\colon P \to P'$ be a* PG *morphism where $P = (G, N)$ and $P' = (G', N')$. For $k \geq 0$, we say that ψ is a* k-monomorphism *if for all $x, y \in E_G \cup V_G \cup T_N$*

(i) $depth(x) < k \wedge \psi(x) = \psi(y) \quad \Rightarrow \quad x = y$;
(ii) $depth(x) < k \Rightarrow depth(x) = depth(\psi(x))$, *i.e., ψ preserves depth information up to k.*

We say that ψ is a k-isomorphism *if it is a k-monomorphism and additionally it is surjective on items of depth smaller than k, i.e.,*

(iii) $\forall x' \in E_{G'} \cup V_{G'} \cup T_{N'}.\ (depth(x') < k \ \Rightarrow\ \exists x \in E_G \cup V_G \cup T_N.\ \psi(x) = x')$.

A marked Petri graph morphism $\psi\colon (P, u) \to (P', u')$ is a k-monomorphism (k-isomorphism, respectively) if so is its Petri graph component $\psi\colon P \to P'$.

Every Petri graph morphism is trivially a 0-isomorphism, hence this holds also for the only morphism from the Petri graph in Fig. 10(a) to that in Fig. 10(b) (mapping transitions t_1 and t_2 of the first Petri graph to the corresponding transitions in the second one). However, it is not a 1-monomorphism since it "merges" two A-edges whose depths are 0 and 1, respectively.

The next lemma shows that for surjective PG morphisms the conditions of Definition 28 can be greatly simplified. Then we make explicit some closure properties of k-monomorphisms and k-isomorphisms.

Lemma 29 *Let $k \geq 0$ and let $\psi\colon P \to P'$ be a surjective* PG *morphism satisfying Condition (i) in Definition 28. Then ψ is a k-isomorphism.*

PROOF. See the Appendix. □

Lemma 30 *The class of k-monomorphisms and the class of k-isomorphisms are closed under composition. If $\psi_1 \circ \psi_2$ is a k-monomorphism, then ψ_2 is a k-monomorphism. And furthermore if $\psi_1 \circ \psi_2$ is a k-isomorphism and ψ_2 satisfies Condition (iii), then both ψ_1 and ψ_2 are k-isomorphisms.*

PROOF. Straightforward. □

As anticipated above, the relevance of the classes of morphisms just introduced lies in the fact that given a $k \geq 0$, for each $i \geq 0$ the morphism $\psi_i : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ relating the Petri graphs generated by the algorithm for k-covering of Definition 26 are k-monomorphisms. Conceptually, this means that once an item is generated at depth smaller than k, its depth does not change later. Furthermore, from a certain point on, when all items of depth smaller than k have already been generated, the above morphisms are also k-isomorphisms.

Lemma 31 *For a fixed $k \geq 0$, the PG_ι morphisms $\psi_i : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ from Definition 21 and Definition 26 are k-monomorphisms.*

PROOF. See the Appendix. □

4.2.1 Termination

The fact that the algorithm computing the k-covering always terminates is not obvious. In the proof of termination, a crucial role is played by Lemma 34 below, stating that it is not possible to perform infinitely many unfolding steps, without having the folding condition (F3) satisfied at some stage. This lemma is actually independent of the graphical structure and can be proved by considering only the causality structure of a Petri graph, as expressed by the underlying pre-net. More formally, we show that in any infinite pre-net, satisfying suitable acyclicity and well-foundedness requirements, there exists a pair of transitions t, t' (called a *folding pair*) such that the pre-set of t' is dependent on t in the sense of Condition (F3) of Definition 26. Let us start by formalising the notion of folding pair.

Definition 32 (folding pair) *Let $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ be a pre-net. A* folding pair *in N is a pair of transitions $t, t' \in T$ such that $t \neq t'$, $p(t) = p(t')$, $|{}^\bullet t| = |{}^\bullet t'|$, $|\underline{t}| = |\underline{t'}|$, and for all $1 \leq j \leq |{}^\bullet t| + |\underline{t}|$ it holds that either $[{}^\bullet t \cdot \underline{t}]_j = [{}^\bullet t' \cdot \underline{t'}]_j$ or $t <_N [{}^\bullet t' \cdot \underline{t'}]_j$.*

We will also need an operation which removes from a given pre-net a subnet and all its consequences, as expressed by the following definition. Recall that a *backward conflict* in a (pre-)net N occurs if there is a place s which belongs to the post-set of more than one transition, i.e., $s \in t^\bullet \cap t'^\bullet$ for two transitions $t \neq t'$.

Definition 33 *Let $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ be an acyclic pre-net without backward conflicts and let $Q \subseteq T$. We define*

$$N - Q = (S', T', {}^\bullet(\ )_{|T'}, (\ )^\bullet{}_{|T'}, \underline{(\ )}_{|T'}, p_{|T'}),$$

where $S' = S - \{s \mid \exists t \in Q\colon t <_N s\}$ *and* $T' = T - \{t' \mid \exists t \in Q\colon (t <_N t' \vee t = t')\}$*, i.e., all elements of* Q *and their consequences are removed from* N*.*

The next key lemma ensures that in any infinite net obtained by applying unfolding steps only, there exists a folding pair. More specifically, it shows that given a net N of this kind, if we consider an infinite set of transitions Q with the same label, either it contains a folding pair or we can remove almost all (namely all but finitely many) elements of Q from N in a way that the resulting net remains infinite. Using this fact if N did not contain any folding pair we would reach a contradiction: since the set of labels is finite, we could remove in a finite number of steps almost all the transitions of N, still getting at the end an infinite net.

Lemma 34 *Let* $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ *be an infinite irredundant pre-net, labelled over a finite set* A*, and satisfying the following conditions:*

(i) *the relation* $<_N$ *is acyclic;*
(ii) *there are no backward conflicts;*
(iii) *for any* $x \in S \cup T$ *the set* $\lfloor x \rfloor$ *(the causes of* x*) is finite;*
(iv) *the set* $Min(N) = \{s \mid \lfloor s \rfloor = \emptyset\}$ *is finite, i.e., only finitely many places have an empty set of causes;*
(v) *for* $t, t' \in T$ *with* $p(t) = p(t')$ *it holds that* $|{}^\bullet t| = |{}^\bullet t'|$ *and* $|\underline{t}| = |\underline{t'}|$*.*

Let $Q \subseteq T$ *be a set of transitions with the same action label, i.e.,* $\exists a \in A. \forall t \in Q.\ p(t) = a$*, such that* Q *does not contain a folding pair. Then there exists a set* $Q' \subseteq Q$ *such that* $Q - Q'$ *is finite and* $N - Q'$ *is infinite.*

PROOF. See the Appendix. □

We can now show that there cannot be an infinite net without a folding pair.

Lemma 35 *If* $N = (S, T, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p)$ *is a pre-net satisfying the conditions of Lemma 34, then it contains a folding pair.*

PROOF. Let $A' = \{a \in A \mid \exists^\omega t\colon p(t) = a\}$, i.e., the set of all action labels that occur infinitely often in the net. The set A' is obviously finite, since A is finite. Hence the proof can proceed by induction on $|A'|$.

- If $|A'| = 0$, then N is finite and the lemma is trivially true, since the premise of the implication is false.
- Assume that the lemma holds for $|A'| = k$ and consider the case $|A'| = k+1$. Consider any $a \in A'$ and the set $Q_a = \{t \in T \mid p(t) = a\}$. Then according to Lemma 34, either Q_a contains a folding pair and we are done, or we can

remove almost all the elements of Q_a, obtaining an infinite net N'. Since in N' only k action labels occur infinitely often, but N' is still infinite, by induction hypothesis, it follows that N' contains a folding pair, which is also a folding pair of N. $\Box$

The above lemma ensures that in our algorithm a folding step will be eventually performed. We are now ready to show termination of the algorithm. We will use the notion of marked pre-net morphism, i.e., the net component of a Petri graph morphism. Explicitly, a marked pre-net morphism $\beta : (N, m) \to (N', m')$ is a pair of functions $\beta = \langle \beta_S : S \to S', \beta_T : T \to T' \rangle$ such that ${\beta_S}^*(m) = m'$ and for any $t \in T$, ${\beta_S}^*({}^\bullet t) = {}^\bullet \beta_T(t)$, ${\beta_S}^*(\underline{t}) = \underline{\beta_T(t)}$ and ${\beta_S}^*(t^\bullet) = \beta_T(t)^\bullet$.

Proposition 36 (termination) *The algorithm computing the k-covering (see Definition 26) terminates for every graph grammar $\mathcal{G}$ and every $k \in \mathbb{N}$.*

PROOF. (Sketch - See the Appendix for full proof) In parallel to the computation of the k-covering $\mathcal{C}^k(\mathcal{G})$ we construct a second *acyclic* pre-net N' as follows. For every unfolding step, we extend N' with a new transition, corresponding to the transition added in $\mathcal{C}^k(\mathcal{G})$. Instead N' is left unchanged in a folding step. The construction ensures the existence of a surjective net morphism from N' to its "folded" counterpart, i.e., the pre-net underlying the Petri graph constructed by the algorithm.

Suppose by contradiction that the algorithm does not terminate and thus that N' grows without bounds. Enrich the labels of the transitions in N' by adding, besides the name of the corresponding rewriting rule, also the edges of depth smaller than k occurring in the pre-set and in the context of the corresponding transition of the k-covering, and the nodes attached to these edges. Since there are only finitely many items of depth smaller than k, there will be finitely many such labels.

Thus, by Lemma 35, N' contains a folding pair $\hat{u}$, $\hat{t}$. The image of such a folding pair through the net morphism from N' to the pre-net underlying $\mathcal{C}^k(\mathcal{G})$ provides a folding pair u, t in $\mathcal{C}^k(\mathcal{G})$. The way in which the transition labels in N' have been enriched allows to show that the second transition t in the pair could never have been added to the Petri graph since this would have been a violation of the third condition of the unfolding step in Definition 26. $\Box$

4.2.2 Confluence

In order to prove that the algorithm produces a uniquely determined result, independently of the order in which folding and unfolding steps are applied, we show that the rewriting relation on Petri graphs induced by folding and unfolding steps is locally confluent.

We first need two preliminary lemmata. The first one observes that the coverability property of markings is preserved under pre-net morphisms (and thus also under Petri graph morphism).

Lemma 37 *Let $(N, m_N), (N', m_{N'})$ be marked pre-nets and let $\beta : N \to N'$ be a marked pre-net morphism. If a marking m is coverable in (N, m_N), then $\beta^{\oplus}(m)$ is coverable in $(N', m_{N'})$.*

PROOF. Trivial, since pre-net morphisms are simulations. □

The second lemma shows that, under specific conditions, folding and unfolding steps have no effect on the Petri graph. However, notice that in both these cases the corresponding application conditions would not be satisfied.

Lemma 38 *Let (P, u) with $P = (G, N)$ be a Petri graph for a graph grammar $\mathcal{G}$, let $r = (L \hookleftarrow K \hookrightarrow R)$ be a rewriting rule of $\mathcal{G}$ and let $\psi : L \to G$ be a match of the left-hand side in G. Then $\mathsf{fold}((P, u), r, \psi, \psi) \cong (P, u)$.*

If furthermore P contains a transition $t \in T_N$ such that $p_N(t) = r$ and ${}^\bullet t = \psi^*(\lambda(E_L - E_K))$ and $\underline{t} = \psi^*(\lambda(E_K))$, then $\mathsf{unf}((P, u), r, \psi) \cong (P, u)$.

PROOF. Immediate. Only note that the second part of the lemma, concerning unfoldings, requires the irredundancy condition: The newly unfolded transition is immediately merged with t. □

We fix some notational conventions. We write $(P, u) \rightsquigarrow^{unf}_{r,\psi} (P', u')$ whenever $(P', u') \cong \mathsf{unf}((P, u), r, \psi)$. We write $(P, u) \dashrightarrow^{unf}_{r,\psi} (P', u')$ whenever $(P, u) \rightsquigarrow^{unf}_{r,\psi} (P', u')$ and the application conditions (U1)–(U3) of the unfolding step are satisfied. In the same way $(P, u) \rightsquigarrow^{fold}_{r,\psi,\eta} (P', u')$ whenever $(P', u') \cong \mathsf{fold}((P, u), r, \psi, \eta)$, and again $(P, u) \dashrightarrow^{fold}_{r,\psi,\eta} (P', u')$ whenever $(P, u) \rightsquigarrow^{fold}_{r,\psi,\eta} (P', u')$ and the application conditions (F1)–(F4) of the folding step are satisfied.

Furthermore we write $(P,u) \dashrightarrow (P',u')$ whenever $(P,u) \dashrightarrow^{unf}_{r,\psi} (P',u')$ or $(P,u) \dashrightarrow^{fold}_{r,\psi,\eta} (P',u')$, for suitable rule r and morphisms ψ, η.

We are now ready to prove the confluence result.

Proposition 39 (local confluence) *Let $(P,u) \dashrightarrow (P_i,u_i)$ for $i \in \{1,2\}$. Then there is a Petri graph (P',u') such that $(P_i,u_i) \dashrightarrow^* (P',u')$ for $i \in \{1,2\}$.*

PROOF. (Sketch - See the Appendix for full proof) The proof mainly relies on the fact that both folding and unfolding operations can be described as colimits in a suitable category of Petri graphs. Then a general categorical result that ensures the commutativity of colimits can be exploited. Finally, things must be accommodated to take into account also the applicability conditions of folding and unfolding steps as described in the algorithm (see Definition 26). □

For instance, in Fig. 12 the reader can find two confluent runs of the algorithm computing the 0-covering of the running example grammar $\mathcal{G}$.

A general result ensures that for a rewriting system, local confluence and termination imply confluence. Hence we immediately get the following result.

Proposition 40 (confluence) *For any grammar $\mathcal{G}$ and $k \in \mathbb{N}$ the algorithm computing the k-covering terminates with a result $\mathcal{C}^k(\mathcal{G})$, unique up to isomorphism.*

PROOF. Recall that, by Proposition 39, the algorithm computing the k-covering is confluent and, by Proposition 36, it is also terminating. Then uniqueness follows from the Newman's Lemma [19], which states that local confluence and termination imply global confluence. □

We can also show that the approximated unfolding algorithm produces a unique result, even if we violate Condition (U3) of the unfolding step a fixed number of times. This will be useful for constructing morphisms between the various k-truncations and k-coverings.

Lemma 41 *If Condition (U3) is violated a finite number of times during the construction of the covering, the algorithm of Definition 26 still terminates with the same unique result.*

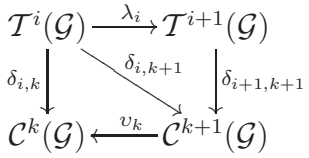
PROOF. See the Appendix. □

4.2.3 Over-approximation

We first show that the computed Petri graph $\mathcal{C}^k(\mathcal{G})$ provides an over-approximation of the behaviour of the given graph grammar, which is exact up to causal depth k. More precisely we prove that from any graph reachable in $\mathcal{G}$, there is a morphism into the covering $\mathcal{C}^k(\mathcal{G})$ such that the image of its edge set corresponds to a reachable marking. Furthermore, if a graph is reachable in $\mathcal{G}$ in less than k steps, then it can be mapped injectively to (the graphical component of) $\mathcal{C}^k(\mathcal{G})$.

Rather than proving this result directly, we will show that there is a k-isomorphism from the full unfolding $\mathcal{U}(\mathcal{G})$ into each covering $\mathcal{C}^k(\mathcal{G})$. In order to obtain this result and some further results in Section 4.3, we will now first define morphisms $\upsilon_k \colon \mathcal{C}^{k+1}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and $\delta_{i,k} \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$.

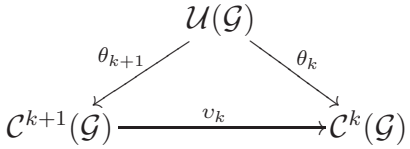
Proposition 42 (k-Monomorphisms υ_k and $\delta_{i,k}$) *For any $k, i \in \mathbb{N}$ there are k-monomorphisms $\upsilon_k \colon \mathcal{C}^{k+1}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and $\delta_{i,k} \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ such that the following diagram commutes, where the morphisms $\lambda_i \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{T}^{i+1}(\mathcal{G})$ are those forming the truncation tower, as introduced in Definition 22. Furthermore, the morphisms υ_k and $\delta_{i,k}$ for $i \geq k$ are k-isomorphisms.*



PROOF. See the Appendix. □

We are now ready to show that there is a k-isomorphism from the full unfolding into every covering $\mathcal{C}^k(\mathcal{G})$. This will be used to prove that the k-covering represents in an exact way the behaviour of the grammar up to causal depth k.

Proposition 43 *For every index k there is a marked* PG *morphism which is a k-isomorphism $\theta_k \colon \mathcal{U}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and such that the following diagram commutes.*



PROOF. See the Appendix. □

The correctness of the algorithm, specifically the fact that it computes an over-approximation or abstraction of the original system that is exact up to depth k, now follows as a corollary of the last proposition.

Corollary 44 (over-approximation) *Let $\mathcal{G} = (\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar and assume that the algorithm computing the k-covering terminates producing the Petri graph $\mathcal{C}^k(\mathcal{G}) = ((U, N), u)$. Let $\mathbb{G}$ be the set of graphs reachable in $\mathcal{G}$, let $\mathbb{M}$ be the set of markings of N reachable from the marking $m_0 = \mathbf{m}(u)$.*

Then there exists a relation $\mathcal{S} \subseteq \mathbb{G} \times \mathbb{M}$, satisfying the following properties:

(i) $(G_{\mathcal{R}}, m_0) \in \mathcal{S}$.
(ii) *$\mathcal{S}$ is a simulation, i.e., if $(G, m) \in \mathcal{S}$ and $G \Rightarrow_r G'$ for some rule $r \in \mathcal{R}$, then $m\,[t\rangle\, m'$ where $p_N(t) = r$ and $(G', m') \in \mathcal{S}$.*
(iii) *For every pair $(G, m) \in \mathcal{S}$ there exists a morphism $\varphi_G \colon G \to U$ such that ${\varphi_G}^{\oplus}(E_G) = m$. If $G_{\mathcal{R}} \Rightarrow^*_{\mathcal{R}} G$ with a derivation of length smaller than k, then φ_G is injective.*
(iv) *If $G_{\mathcal{R}} \Rightarrow^*_{\mathcal{R}} G$ with a derivation of length smaller than k and furthermore $(G, m) \in \mathcal{S}$ and $m\,[t\rangle\, m'$, then $G \Rightarrow^*_r G'$ for some rule $r \in \mathcal{R}$ where $p_N(t) = r$ and $(G', m') \in \mathcal{S}$.*

PROOF. By Proposition 43 there is a marked Petri graph morphism $\theta_k \colon \mathcal{U}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and thus there is obviously a simulation between the markings of $\mathcal{U}(\mathcal{G})$ and the markings of $\mathcal{C}^k(\mathcal{G})$. Since θ_k is a k-isomorphism, this simulation is a bisimulation for markings which can be reached in less than k derivation steps.

Recall from Proposition 24 that the full unfolding $\mathcal{U}(\mathcal{G})$ provides an "exact" representation of the behaviour of $\mathcal{G}$, in the sense that there is a bisimulation relating the reachable graphs of $\mathcal{G}$ and the reachable markings of $\mathcal{U}(\mathcal{G})$. Hence the result immediately follows. □

Observe that the existence of a morphism $\theta_k : \mathcal{U}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ for any k (see Proposition 43) allows also to trivially deduce that causality in each covering $\mathcal{C}^k(\mathcal{G})$ provides an approximation of causality in the full unfolding. More precisely, for any k, if $\mathcal{U}(\mathcal{G}) = ((U, N), u)$ and $\mathcal{C}^k(\mathcal{G}) = ((U_k, N_k), u_k)$ then for all $x, y \in E_U \cup T_U$, if $x <_N y$ then $\theta_k(x) <_{N_k} \theta_k(y)$.

4.3 Full unfolding as limit of the coverings

Intuitively, the sequence of k-coverings $\mathcal{C}^k(\mathcal{G})$, with growing k, represents the behaviour of the graph grammar $\mathcal{G}$ with arbitrary high accuracy. This is formalised by showing that the sequence of k-coverings $(\mathcal{C}^k(\mathcal{G}))_{k\in\mathbb{N}}$ converges to the full unfolding $\mathcal{U}(\mathcal{G})$, or, in categorical terms, that $\mathcal{U}(\mathcal{G})$ is the limit of the chain $(\mathcal{C}^k(\mathcal{G}))_{k\in\mathbb{N}}$ in the category PG_ι of marked Petri graphs.

Using the morphisms constructed in Section 4.2, we define the notion of covering tower, which is the counterpart of the truncation tower.

Definition 45 (covering tower) *The following diagram, where the υ_k are defined as in Proposition 42,*

$$\mathcal{C}^0(\mathcal{G}) \overset{\upsilon_0}{\leftarrow} \ldots \mathcal{C}^k(\mathcal{G}) \overset{\upsilon_k}{\leftarrow} \mathcal{C}^{k+1}(\mathcal{G}) \overset{\upsilon_{k+1}}{\leftarrow} \ldots$$

is called the covering tower*.*

Now we can show that the full unfolding can be obtained as the limit of the covering tower.

Proposition 46 (unfolding as limit of the coverings) *The limit in the category PG_ι of the covering tower $\mathcal{C}^0(\mathcal{G}) \overset{\upsilon_0}{\leftarrow} \ldots \mathcal{C}^k(\mathcal{G}) \overset{\upsilon_k}{\leftarrow} \mathcal{C}^{k+1}(\mathcal{G}) \overset{\upsilon_{k+1}}{\leftarrow} \ldots$ is the full unfolding $\mathcal{U}(\mathcal{G})$ of the graph grammar.*

PROOF. See the Appendix. □

5 Applications

We will now discuss how truncations (under-approximations) and coverings (over-approximations) can be used for verification purposes. We will illustrate this with several examples and give a short introduction to available tool support.

5.1 Verification

In Sections 3 and 4 we constructed k-truncations $\mathcal{T}^k(\mathcal{G})$ and k-coverings $\mathcal{C}^k(\mathcal{G})$ for a given graph transformation system $\mathcal{G}$. The former have a Petri graph morphism into the full unfolding $\mathcal{U}(\mathcal{G})$ whereas for the latter there is a morphism from the full unfolding. The situation is summarized in Figure 13. Note also

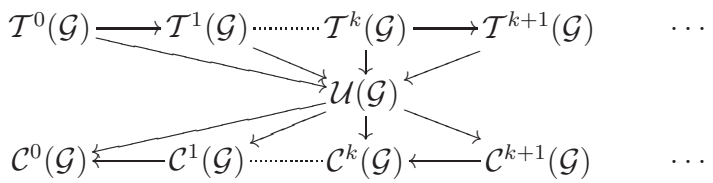


Fig. 13. Unfolding as the colimit of truncations and limit of coverings.

that $\mathcal{U}(\mathcal{G})$ is the colimit of the chain of truncations and the limit of the chain of coverings.

There are two ways in which we can exploit the existence of these morphisms for verification purposes: in order to obtain simulations and in order to approximate causality.

Petri graph morphisms are simulations, i.e., a step of the source Petri graph can always be simulated by the target. Then, a general issue when model checking approximated transition systems is that usually only fragments of the temporal logic CTL* such as ACTL or LTL can be considered [15]. In general, properties involving existential quantifications over computations, like "there exists a path such that ...", cannot be checked for over-approximations, since spurious runs, not appearing in the original system, may be introduced by the approximation. Similarly, properties involving universal quantification, like "for all paths it holds that ...", cannot be checked on under-approximations.

A second issue is concerned with the fact that truncations and coverings approximate, not only the transition system of the original grammar, but also its states: graphs reachable in the original grammar are represented by markings in the Petri graphs. Consequently, in order to be able to transfer verification results from one transition system to the other, some restrictions will need to be imposed on the considered state properties.

The situation can hence be summarized as follows:

Proposition 47 (simulation) *Let $\psi\colon (P,u) \to (P',u')$ be a morphism between marked Petri graphs. Then there is a relation R on the reachable markings of P and P' which is a simulation with respect to the firing relation of the underlying nets.*

Furthermore if $(P,u) = \mathcal{T}^k(\mathcal{G})$ and $(P',u') = \mathcal{U}(\mathcal{G})$ for some graph transformation system $\mathcal{G}$, then there is a simulation relation R such that for every $(m,m') \in R$ there exists a bijective graph morphism $\varphi\colon graph_G(m) \to graph_{G'}(m')$.

Similarly if $(P,u) = \mathcal{U}(\mathcal{G})$ and $(P',u') = \mathcal{C}^k(\mathcal{G})$ for some graph transformation

system $\mathcal{G}$, then there is a simulation relation R such that for every $(m, m') \in R$ there exists an edge-bijective graph morphism φ: $graph_G(m) \to graph_{G'}(m')$.

Note that for truncations we obtain bijective morphisms, whereas for coverings we get only edge-bijective morphisms due to the folding steps and the resulting node fusions. Furthermore the set of all graphs generated from the reachable markings of the full unfolding corresponds exactly to the set of reachable graphs of $\mathcal{G}$ (up to isolated nodes). From this we can conclude that whenever we have a property on graphs that is reflected by edge-bijective graph morphisms and which holds for all graphs generated by markings reachable on the covering, then it will also hold for all graphs reachable in the original graph transformation system. A typical example for a property that is reflected by edge-bijective morphisms is non-adjecency of edges or the absence of certain paths or cycles. That is, for a graph morphism $\varphi: G \to G'$, whenever G' does not contain such a forbidden structure, then this is also true for G. Additionally we can transfer information on the number of edges with a certain label due to edge-bijectivity.

In [10,11,7] we have shown how to formulate and verify reachability properties and other properties using temporal logics and second-order monadic logic on graphs. Furthermore [10] also explains how to view our technique as a specific instance of abstract interpretation [39].

A second, different use of Petri graph morphisms is that they over-approximate causality. That is, if there is no causal relation between certain transitions on the over-approximation, then there is no causal dependence between the corresponding rule applications in the graph transformation system. This is explained in more detail in Example 2 below.

5.2 Examples

Based on these ideas we now present some examples taken from the area of mobile systems and processes. These are simple examples meant to clarify the concepts introduced earlier. More complex examples and case studies have been conducted with the help of the tool support presented in Section 5.3.

Example 1: Consider the simple graph grammar $\mathcal{S}$ in Fig. 14, where edge labels have the following meaning: C (connections), S_{pub} (public servers), S_{prv} (private servers), P_{int} (internal processes) and P_{ext} (external processes). Furthermore edges G_{pub} and G_{prv} are intended to represent generators, which produce public and private servers, respectively. The rules of the grammar allow to generate an arbitrary network of servers connected by C-edges, with

the only restriction that no connections are allowed from a public to a private server. Connections in the other direction are admissible.

Furthermore servers may create processes—public servers create external processes and private servers create internal processes—and processes are mobile and can wander around in the network. Observe that in the rule describing a process crossing a connection, the C-edge is in the context (denoted by a dashed line), i.e., it is preserved, and not first deleted and then created again. We want to show that an external process will never be connected to a private server, thus accessing classified data. It can easily be seen that this property, which talks about non-adjacency of edges, is reflected by (edge-bijective) graph morphisms.

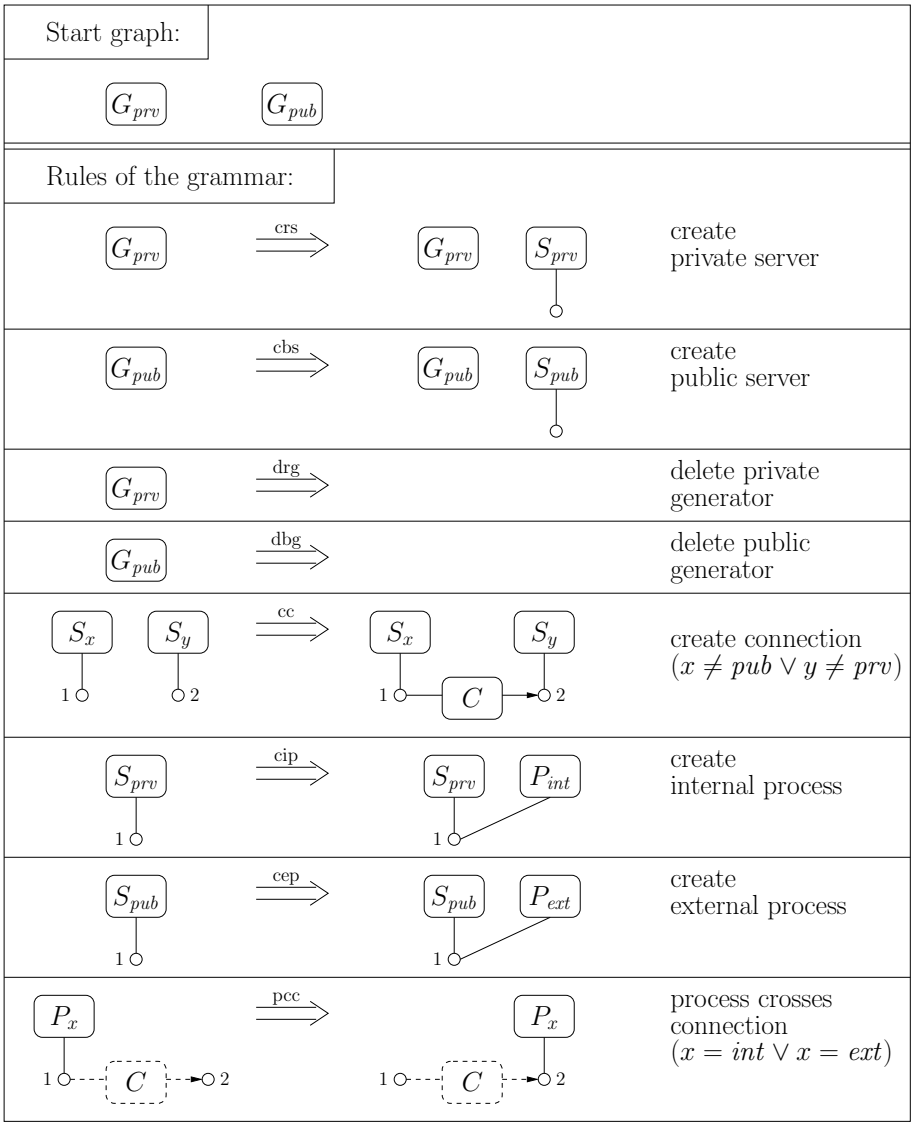


Fig. 14. The example graph grammar $\mathcal{S}$.

The algorithm in Definition 26, applied to the graph grammar $\mathcal{S}$ in order to

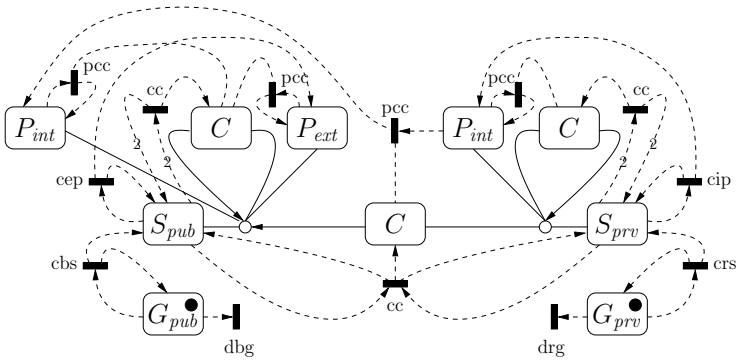


Fig. 15. The 0-covering $\mathcal{C}^0(\mathcal{S})$ of grammar $\mathcal{S}$ in Fig. 14.

compute the 0-covering $\mathcal{C}^0(\mathcal{S})$, produces the Petri graph in Fig. 15. Observe that transitions are annotated with the corresponding rule names. Inspecting $\mathcal{C}^0(\mathcal{S})$ we can establish that the above mentioned property holds by simply examining the graph structure underlying the 0-covering, since edges of the form S_{prv} and of the form P_{ext} do not share a common node.

Example 2: In order to show that, to be able to prove a property of interest, it might be useful to compute the k-covering for some k strictly greater than 0, we consider a simple graph grammar $\mathcal{S}'$ in Fig. 16, related to the one analysed in the previous example. Edges represent either servers (S), or mobile processes (P, Q), or connections (C). It contains a rule for process creation, similar to the one in Fig. 14. Processes may cross connections and may also change their state (from P to Q and back).

The 0-covering of $\mathcal{S}'$ is the Petri graph $\mathcal{C}^0(\mathcal{S}')$ in Fig. 17. Note that the two distinct nodes of the start graph are fused in the approximation. Using $\mathcal{C}^0(\mathcal{S}')$ we cannot prove a property such as "no two servers are ever attached to the same port", although this property holds for the original system. Another property, satisfied by $\mathcal{S}'$, but not verifiable in $\mathcal{C}^0(\mathcal{S}')$ is the fact that processes created by the right-hand server may not cross the connection.

Computing the 1-covering we obtain the Petri graph $\mathcal{C}^1(\mathcal{S}')$ in Fig. 18, where each edge, node and transition is decorated with a number representing its depth. Now, using $\mathcal{C}^1(\mathcal{S}')$ we can first show that indeed no two servers are attached to the same port. This is not directly visible in the underlying graph since in fact two hyperedges labelled S are attached to the rightmost node, but it can be proved by observing that there is no reachable marking that covers them both at the same time. Secondly, recalling that the causality in $\mathcal{C}^1(\mathcal{S}')$ approximates the causality in the original system, we can conclude that there is no causal dependency between a process on the right and a process on the left, and thus that processes on the right will never cross the connection.

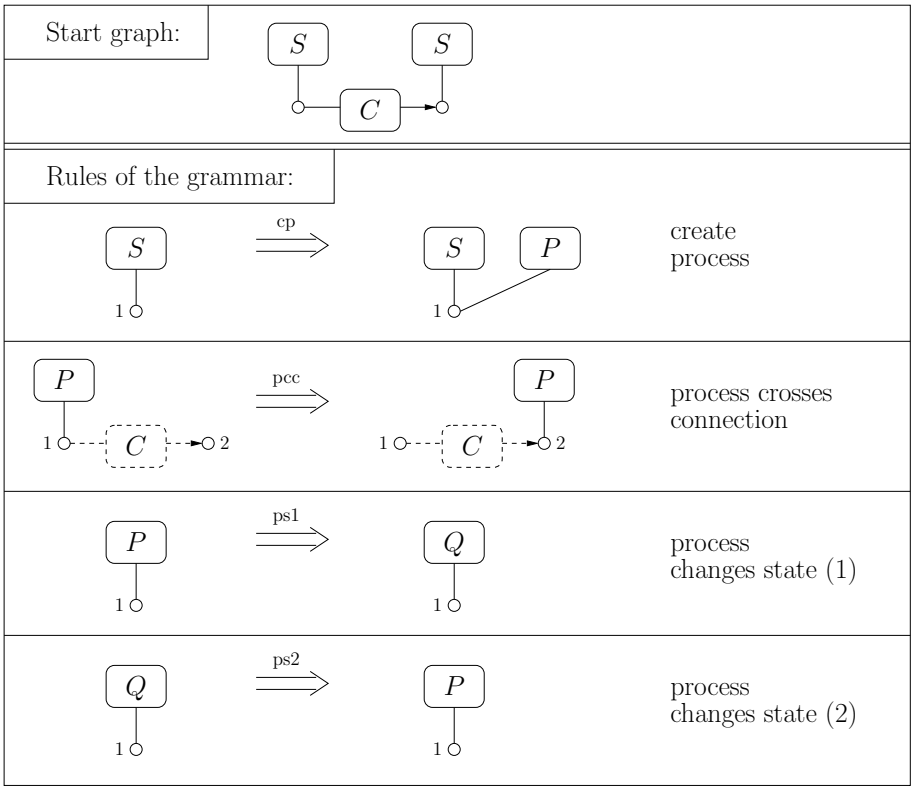


Fig. 16. Graph transformation system $\mathcal{S}'$ with process state change.

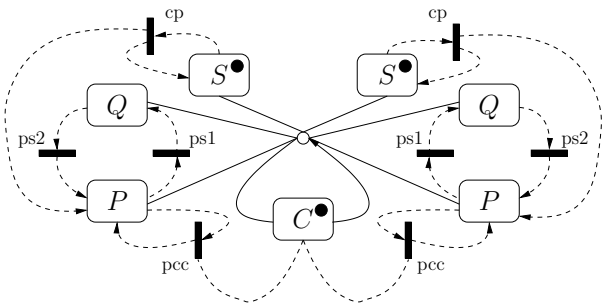


Fig. 17. The 0-covering $\mathcal{C}^0(\mathcal{S}')$ of grammar $\mathcal{S}'$ in Fig. 16.

Example 3: In the last example we show how contextual arcs can help to decrease the size of an unfolding. Consider the graph grammar $\mathcal{S}''$ in Fig. 19. It contains only one rule describing processes crossing connections. The rule comes in two variants: either the connection is in the context, i.e., it is preserved (variant A), or it is deleted and recreated (variant B). If we compute the 0-coverings of the two variants, we obtain the Petri graphs in Fig. 20. Observe that the number of transitions in the left-hand Petri graph (variant A) coincides with the number of processes waiting to cross the connection. Instead, in variant B, we obtain a combinatorial explosion since in this case the order of crossings is relevant: either the first process crosses first and then the second, or vice versa, and the two situations are represented in different branches of the unfolding. In general, for n processes, variant A would lead

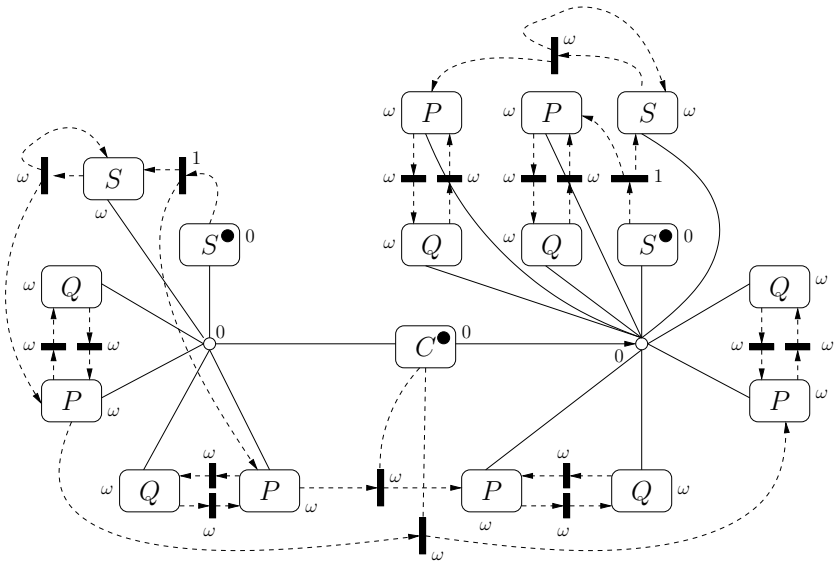


Fig. 18. The 1-covering $\mathcal{C}^1(\mathcal{S}')$ of grammar $\mathcal{S}'$ in Fig. 16.

to n transitions, whereas variant B would produce $n + n \cdot (n-1) + \cdots + n!$ transitions.

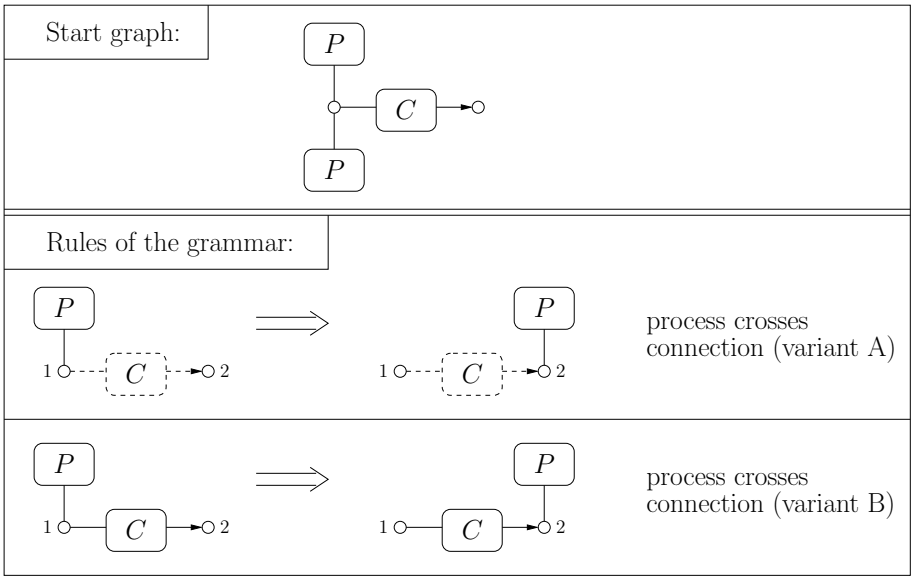


Fig. 19. Graph transformation system $\mathcal{S}''$.

5.3 Tool Support

Although some parts of this article are fairly abstract and theoretical in nature, the application of our results to system verification is quite immediate. We have implemented the algorithm computing the k-covering of a given graph transformation system. The tool—called AUGUR—can be downloaded at `http://www.ti.inf.uni-due.de/research/augur_1/` (see also [33,36]).

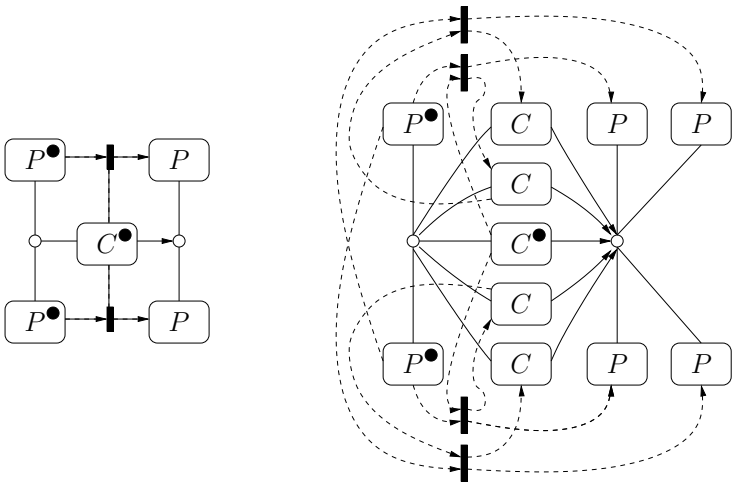


Fig. 20. The 0-covering of grammar $\mathcal{S}''$ in Fig. 19 in variant A (left) and variant B (right).

The input and output of AUGUR is in GXL and GTXL, which are XML standards for the exchange of graphs and graph transformation systems, respectively. Suitable converters have been added in order to visualise rules and Petri graphs and to extract the Petri net component of a Petri graph, which can then be used as input for a Petri net analysis tool such as LoLA [55] and other tools which are included in the implementation. We have used this implementation in order to conduct case studies, analysing dynamic data structures such as red-black trees [3], properties of communication protocols such as mutual exclusion [21] and absence of deadlocks. Furthermore we have studied a simplified network with a firewall [5].

The current implementation supports only rewriting rules with discrete interfaces (i.e., edges cannot be preserved). An implementation dealing with general rules, and thus fully supporting the theory in this paper, is under development. Actually, observe that general rules which preserve edges can be simulated by rules which delete and recreate the same edges. As already discussed, this encoding does not alter the set of reachable graphs, but it could possibly reduce the concurrency in the system and thus lead to the generation of a larger covering (as shown in Example 3 of Section 5.2). Unfortunately, the reduction of the level of concurrency in the system, with the insertion of new causal dependencies, can also enable additional folding steps which can affect the provability of certain properties.

For example, let us consider the graph grammar $\mathcal{S}$ in Fig. 14: in order to feed it into the tool, we changed the last rule in such a way that the edge labelled C is deleted and recreated, instead of being preserved. The 0-covering computed by the tool is shown in Fig. 21 and Fig. 22. The Petri graph is split into a Petri net component (Fig. 21) and a graph component (Fig. 22). As expected, the places of the net (depicted by ovals) and the hyperedges of the

graph (depicted by boxes) coincide.

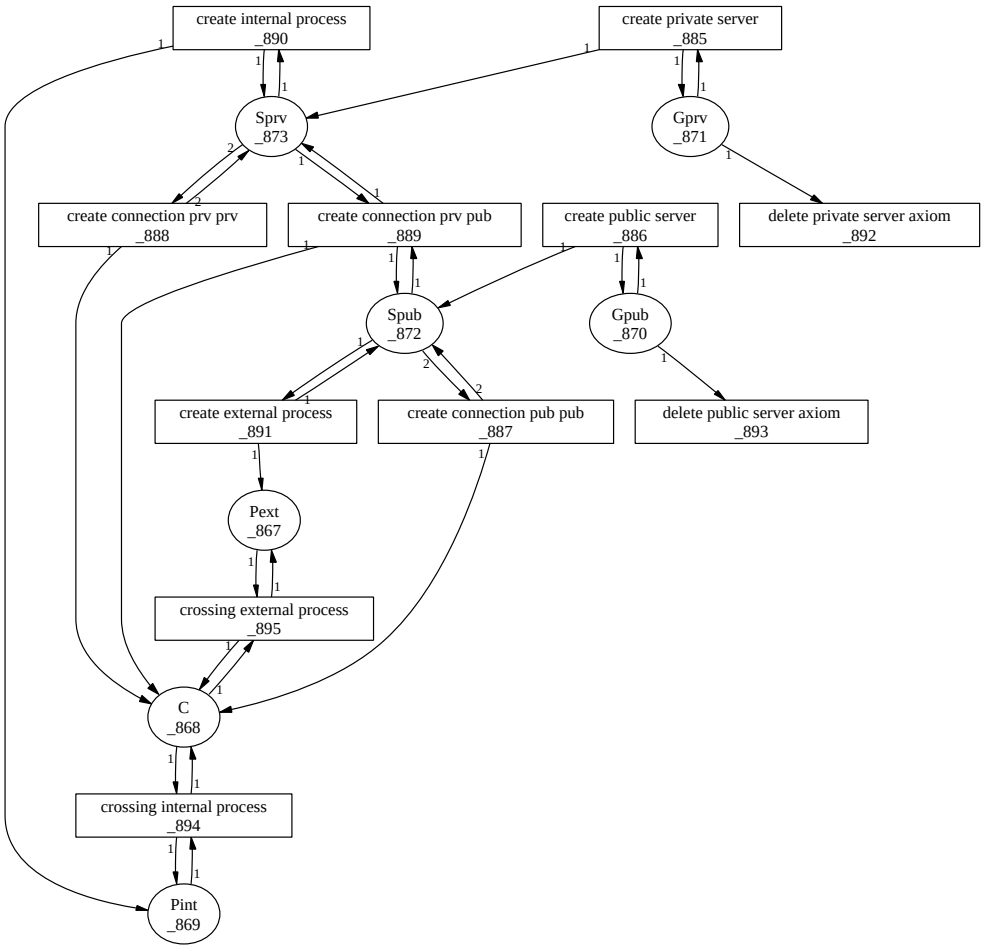


Fig. 21. The Petri net component of the 0-covering (computed by AUGUR).

It is worth observing that unlike in the 0-covering with read arcs depicted in Fig. 15, the property under consideration cannot be verified in the 0-covering of Figures 21 and 22, because of the coarser approximation.

The identification of folding and unfolding pairs requires to establish the coverability of given markings. This can be decided by computing the coverability graph of the Petri net underlying a Petri graph, as described in [46], by incremental coverability checking [35], or alternatively by employing backward reachability [2]. If this gets too costly, the condition of coverability (see Condition (F2) in Definition 26) can be relaxed or checked in an approximated way, a choice which does not compromise the result of correctness (see Corollary 44), but only reduces the "precision" of the algorithm: it will generate a less precise approximation, where less properties of the given GTS can be proved. Indeed AUGUR permits to switch off the coverability checking.

Runtime results concerning test cases with discrete interfaces analysed with

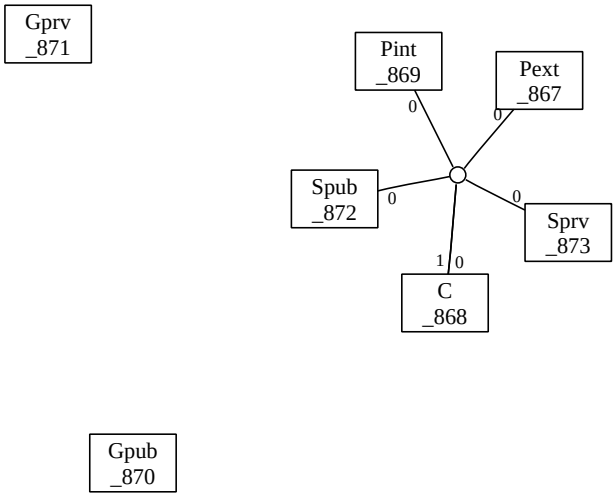


Fig. 22. The graph component of the 0-covering (computed by AUGUR).

AUGUR are presented in [34,37]. Unfortunately, for the examples of the present paper performance results are not available, as they heavily rely on non-discrete interfaces which, as mentioned above, are not yet implemented in AUGUR.

6 Related Work and Conclusions

We have presented a technique for computing under- and over-approximations of the behaviour of graph transformation systems and we have identified suitable classes of properties of a GTS which can be inferred by analysing its approximations. We envision a scenario where a property of a given GTS can be checked by computing better and better approximations and verifying the property for each of them. Because of undecidability issues, this process might never terminate and it could also be costly from a complexity point of view, but with appropriate heuristics and fine-tuning of the technique, it seems that several interesting properties for non-trivial GTSs can be effectively verified.

Work on verification and analysis of graph transformation systems is rather recent. Roughly, two lines of research can be distinguished: One pursues the idea of translating graph transformation systems into the input language of a model checker [57,20]. However, existing tools usually do not directly support the creation (and deletion) of an arbitrary number of objects while still maintaining a finite state space, making entirely non-trivial their use for checking finite-state GTSs. Similar problems arise for process calculi agents with name creation, which has also led to specialised techniques in this area such as HD-automata [45]. Hence we pursue in this paper an approach which is developed specifically for graph transformation system. The same is true for work presented in [50,48,49,47], which proposes a different approach in which

GTSs are approximated by abstract graph transformation, and not by Petri nets as in our work. The transition system generated by a GTS is abstracted by a so-called abstract graph transition systems, where abstract graphs (or shape graphs) summarize several graph items (nodes and edges) into one item and contain additional summary information concerning node degrees and the number of summarized items. A distinguishing feature of our approach is the fact that it is based on a partial order semantics, a fact that allows to alleviate the state explosion problem typical of the analysis of concurrent systems. Roughly, our approach is expected to be effective for systems with a high degree of concurrency.

Our work also has connections to the area of shape analysis [53], i.e., the analysis of dynamically evolving pointer structures on a program heap. While shape analysis is specialised to data structures—especially linked lists and trees—our approach aims at the analysis of more general graph transformation systems and we do not rely on manually defined predicates and predicate transformers. Furthermore we can derive bounds on the number of occurrences of certain objects, which is, to our knowledge, not possible in existing shape analysis techniques. On the other hand, shape analysis as presented in [53] allows one to obtain fairly precise analysis results concerning reachability and acyclicity in pointer structures, which can not yet be matched by our approach due to its more general nature. In the future we plan to study ways to integrate the 3-valued logic of shape analysis into our approach.

Shape analysis has a technique for abstraction refinement [38], which is based on inductive learning. Also in our work the possibility of computing the k-covering at a chosen level of accuracy already provides a method for abstraction refinement. However when the check over the k-covering, for a given k, fails because of a false negative, passing to the $(k+1)$-covering we refine the abstraction in a way which is independent of the specific counterexample. The paper [34], using counterexample-guided abstraction refinement inspired by [14], shows how to refine only those parts of a covering that contribute to the spurious counterexamples not occurring in the original system. However, in this setting we do not have any convergence result similar to Proposition 46.

Temporal logics—briefly discussed in Section 5.1—can be further developed, in order to have a powerful specification language for graph transformation systems. We have introduced a monadic second-order graph logic in [11], which can be used to specify the interpretation of atomic propositions. We have shown how to translate such formulae into formulae describing Petri net markings. Furthermore we have studied a logic which interleaves temporal operators and first-order quantification [7], which is related to the logics introduced in [59,47].

While in this paper we have considered an approximation where information

concerning the identity of nodes is lost, it seems conceivable to reason not only about *disconnectedness* but also about *connectedness* using a modified approximation. It seems also possible to extend the set of permissible temporal operators by using both the truncations and the coverings at the same time, i.e., by exploiting over- and under-approximations for model-checking, similarly to what is done in [31,17].

Although not strictly related to our work, it is worth mentioning that analysis techniques for GTSs are not restricted to reachability analysis and model checking. Other properties (such as termination and confluence via critical pair analysis) can be analyzed using the AGG tool [56].

A different issue is to develop efficient techniques for the verification of *finite-state* graph transformation systems. In [47,49] it is described how to store and use finite graph transition systems. In [6] we have presented a partial-order method based on finite complete prefixes of an unfolding (see also the remarks at the end of Section 3).

Acknowledgements: We would like to thank Vitali Kozioura for developing and maintaining the tool AUGUR.

A Proofs of results in the paper

Proposition 16 (cocompleteness of Petri graph categories). *Let $\mathcal{R}$ be a* GTS *and $\mathcal{G}$ be a graph grammar. Then the category of Petri graphs $\mathsf{PG}(\mathcal{R})$ and the category of marked Petri graphs $\mathsf{PG}_\iota(\mathcal{G})$ are both cocomplete, i.e., they have all colimits.*

PROOF. (Sketch) Let $\Delta = \langle \mathcal{D}, \mathcal{F} \rangle$ be a diagram in PG, i.e., $\mathcal{D}$ is a graph with nodes $V_\mathcal{D}$ and (binary) arcs $E_\mathcal{D}$, and $\mathcal{F} : \mathcal{D} \to |\mathsf{PG}|$ is a graph morphism from $\mathcal{D}$ to the graph underlying PG. Then Δ has a colimit $\langle P', \{\psi_v : \mathcal{F}(v) \to P'\}_{v \in V_\mathcal{D}} \rangle$ which is obtained as follows.

- The Petri graph P' is the disjoint union of all the Petri graphs in $\mathcal{F}(V_\mathcal{D})$, modulo the least equivalence relation $\equiv$ such that

(i) if x_1 is a node, edge (place) or transition of $\mathcal{F}(v_1)$ and $f : v_1 \to v_2$ is an arc in $\mathcal{D}$, then $x_1 \equiv \mathcal{F}(f)(x_1)$;
(ii) if $t_1 \equiv t_2$ then ${t_1}^\bullet \equiv {t_2}^\bullet$;
(iii) if $e_1 \equiv e_2$ then $c_{G_1}(e_1) \equiv c_{G_2}(e_2)$.
(iv) if $p_{N_1}(t_1) = p_{N_2}(t_2)$, ${}^\bullet t_1 \equiv {}^\bullet t_2$ and $\underline{t_1} \equiv \underline{t_2}$ then $t_1 \equiv t_2$;

where $P_1 = (G_1, N_1)$ and $P_2 = (G_2, N_2)$ are in $\mathcal{F}(V_\mathcal{D})$, and equivalence is extended to sequences in the obvious way.

- For any $v \in V_\mathcal{D}$, the morphism $\psi_v : \mathcal{F}(v) \to P'$ is defined on nodes, edges (places) and transitions as $\psi_v(x) = [x]_\equiv$.

It is easy to verify that the above construction produces a well-defined Petri graph. In particular, conditions (ii)-(iv) above are used to ensure the irredundancy of the resulting Petri graph. The proof of universality is straightforward.

In the marked case, for a diagram Δ as above in category PG_ι the construction of the colimit object (P', u') is identical for the Petri net component P', while the initial state is obtained as $u' = [u_P]_\equiv$ for any $(P, u_P) \in \mathcal{F}(V_\mathcal{D})$. The fact that in (P', u') every edge is coverable and every transition is firable is a consequence of two facts: the morphisms ψ_v for $v \in V_\mathcal{D}$ are *jointly surjective* (i.e., for any x' in P' there exists a $v \in V_\mathcal{D}$ and an $x \in \mathcal{F}(v)$ such that $\psi_v(x) = x'$), and Petri graph morphisms are simulations between the underlying pre-nets, i.e., they preserve firing sequences. □

Lemma 27. *Let $\psi : P \to P'$ be a PG morphism where $P = (G, N)$. Then for any node, edge or transition $x \in V_G \cup E_G \cup T_N$ it holds that $depth(\psi(x)) \geq depth(x)$.*

PROOF. Recall that the function *depth* for a net N is given by $\mathit{fix}(D_N)$, where D_N is as in Definition 19. In turn $\mathit{fix}(D_N) = \bigsqcup\{D_N^n(\mathbf{0}) \mid n \in \mathbb{N}\}$, where $\mathbf{0}$ is the constant function mapping each item to 0.

We first show by induction that $D_N^n(\mathbf{0}) \leq D_{N'}^n(\mathbf{0}) \circ \psi$ for every $n \geq 0$. Clearly this holds for $n = 0$. Now, for any edge $e \in E_G$ we have

$$\begin{aligned}
D_N^{n+1}(\mathbf{0})(e) &= \bigsqcup\{D_N^n(\mathbf{0})(t) \mid e \in t^\bullet\} \\
&\leq \bigsqcup\{D_{N'}^n(\mathbf{0})(\psi(t)) \mid e \in t^\bullet\} \quad \text{(induction hypothesis)} \\
&\leq \bigsqcup\{D_{N'}^n(\mathbf{0})(\psi(t)) \mid \psi(e) \in \psi(t)^\bullet\} \\
&\leq \bigsqcup\{D_{N'}^n(\mathbf{0})(t') \mid \psi(e) \in t'^\bullet\} \\
&= D_{N'}^{n+1}(\mathbf{0})(\psi(e)).
\end{aligned}$$

It follows that $D_N^n(\mathbf{0})(e) \leq (D_{N'}^n(\mathbf{0}) \circ \psi)(e)$ for all $n \geq 0$, which implies $\mathit{depth}(\psi(e)) \geq \mathit{depth}(e)$ for every edge e of the graph underlying P. The corresponding cases for transitions and nodes are handled similarly. □

Lemma 29. *Let $k \geq 0$ and let $\psi\colon P \to P'$ be a surjective* PG *morphism satisfying Condition (i) in Definition 28. Then ψ is a k-isomorphism.*

PROOF. If $k = 0$, then Conditions (ii) and (iii) trivially hold true. Therefore let $k > 0$, assume that $\psi\colon P \to P'$ satisfies Condition (i) in Definition 28, and suppose, by contradiction, that (ii) is not satisfied, i.e., $Z = \{x \in V_G \cup E_G \cup T_N \mid \mathit{depth}(x) < k \ \wedge \ \mathit{depth}(x) < \mathit{depth}(\psi(x))\} \neq \emptyset$. Take $x \in Z$ with minimal depth. If there are edges, transitions and nodes with the same depth, edges and nodes are preferred. Let us show that this leads to a contradiction, by distinguishing the following cases:

- If $\mathit{depth}(x) = 0$, then x is either an edge or a node. If x is an edge, then it is not in the post-set of a transition, but $\psi(x) \in t'^\bullet$ for some t' since $\mathit{depth}(\psi(x)) > 0$. Since ψ is surjective, $t' = \psi(t)$ for some $t \in T_N$. Since $\psi(x) \in \psi(t)^\bullet = \psi^*(t^\bullet)$, there is an edge $e \in t^\bullet$ with $\psi(e) = \psi(x)$. But since $\mathit{depth}(e) > 0$, it holds that $e \neq x$, contradicting Condition (i).

  If x is a node, then, as above, we can conclude that $\psi(x) \in \mu(\psi(t))(V_R \backslash V_K)$ for some transition t, reaching a similar contradiction.
- If $\mathit{depth}(x) = i$ with $0 < i < k$, then x might either be a transition or an edge or a node. If x is a transition, since $\mathit{depth}(\psi(x)) > \mathit{depth}(x) = i$, then there exists an edge $e' \in {}^\bullet\psi(x) \cdot \underline{\psi(x)}$ such that $\mathit{depth}(e') \geq i$. And since $e' \in \psi^*({}^\bullet x \cdot \underline{x})$ there must be an edge $e \in {}^\bullet x \cdot \underline{x}$ such that $\psi(e) = e'$.

Now, since $depth(x) = i$, it holds that $depth(e) \leq i - 1$, which means that $depth(\psi(e)) > depth(e)$. But this contradicts our choice of x in Z.

A similar contradiction arises when x is an edge or a node. □

Lemma 31. *For a fixed $k \geq 0$, the PG_ι morphisms $\psi_i : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ from Definition 21 and Definition 26 are k-monomorphisms.*

PROOF. We distinguish the following cases:

- *Unfolding step*: If the $(i+1)$-st step of the algorithm is an unfolding step, the result easily follows by showing that morphism ψ_i is injective, and, by induction, that it preserves the depth of all items (for those of depth 0, this follows from the preservation of the initial state).
- *Folding step*: If the $(i+1)$-st step of the algorithm is a folding step, it is easily shown that morphism ψ_i is surjective. Then, using Lemma 29, we can show that ψ_i is a k-isomorphism by proving that for each x, y in P_i it holds $depth(x) < k \wedge \psi_i(x) = \psi_i(y) \Rightarrow x = y$.

  In order to obtain a contradiction, let x be an item of P_i of minimal depth $k' < k$ such that there exists a $y \neq x$ with $\psi_i(x) = \psi_i(y)$. This means that $x \equiv y$, where $\equiv$ is the relation induced on the items of P_i by the construction of the coequalizer of $\varphi, \varphi' : [L] \to P$ (see Definition 25), according to Proposition 16. Therefore either there exists an item $z \in [L]$ such that $\varphi(z) = x$ and $\varphi'(z) = y$, or $x \equiv y$ follows by points (ii)-(iv) of the definition of $\equiv$. In the first case, $depth(x) \geq k$ by Condition (F4) of Definition 26, contradicting the assumption. In all the other cases, it is easy to show that there are items $x' \neq y'$ in P_i with $\psi_i(x') = \psi_i(y')$ and $depth(x') < k' = depth(x)$, contradicting the minimality of k'. □

Lemma 34. *Let $N = (S, T, {}^\bullet(\ ), (\ ){}^\bullet, \underline{(\ )}, p)$ be an infinite irredundant pre-net, labelled over a finite set A, and satisfying the following conditions:*

(i) the relation $<_N$ is acyclic;
(ii) there are no backward conflicts;
(iii) for any $x \in S \cup T$ the set $\lfloor x \rfloor$ (the causes of x) is finite;
(iv) the set $Min(N) = \{s \mid \lfloor s \rfloor = \emptyset\}$ is finite, i.e., only finitely many places have an empty set of causes;
(v) for $t, t' \in T$ with $p(t) = p(t')$ it holds that $|{}^\bullet t| = |{}^\bullet t'|$ and $|\underline{t}| = |\underline{t'}|$.

Let $Q \subseteq T$ be a set of transitions with the same action label a, i.e., $\forall t \in Q.\ p(t) = a$, such that Q does not contain a folding pair. Then there exists a set $Q' \subseteq Q$ such that $Q - Q'$ is finite and $N - Q'$ is infinite.

PROOF. Let $Q \subseteq T$ be a subset of transitions as in the hypotheses. If Q is empty, the lemma is trivially true.

Otherwise, let ℓ be the number of places in each pre-set and context of transitions in Q, i.e., $\ell = |{}^\bullet t \cdot \underline{t}|$, for $t \in Q$. The proof proceeds by induction on $n = |I(Q)|$ where $I(Q) = \{i \mid 1 \leq i \leq \ell \ \wedge \ \exists t, t' \in Q.\ [{}^\bullet t \cdot \underline{t}]_i \neq [{}^\bullet t' \cdot \underline{t'}]_i\}$, i.e., the set of indices i for which there are two transitions in Q having different places at position i in their pre-set/context.

$(n = 0)$
In this case, for all $t, t' \in Q$, it holds ${}^\bullet t = {}^\bullet t'$, $\underline{t} = \underline{t'}$ and, by the assumptions on Q, $p(t) = p(t')$. Thus, since N is irredundant, it follows that $t = t'$ for all $t, t' \in Q$, i.e., Q contains at most one element. Since Q is finite we can conclude by setting $Q' = \emptyset$.

$(n > 0)$
Assume that the lemma holds for all sets Q' such that $|I(Q')| < n$ and consider the case $|I(Q)| = n$. Let M be the set of minimal elements of Q with respect to $<_N$. We distinguish the following cases:

- *M is infinite.*
  In this case we consider the set of places in the pre-set and context of transitions in M, namely $S' = \bigcup_{t \in M} \mathbf{m}({}^\bullet t \cdot \underline{t})$.

  We first observe that S' is infinite. In fact, if S' were finite, since all transitions in M have the same label a, by the irredundancy of N we could conclude that M is finite, contradicting the hypothesis.

  Since the transitions in M are minimal, the set S' is still contained in the places of $N - M = N - Q$. Hence $N - Q$ includes infinitely many places. This fact, together with the hypothesis that N has finitely many minimal places, implies the presence of infinitely many transitions in $N - Q$. Therefore $N - Q$ is still infinite and we can set $Q' = Q$.
- *M is finite.*
  We show by induction on $|M|$ that there is a $Q' \subseteq Q$ such that $Q - Q'$ is finite and $N - Q'$ is infinite.

  $(|M| = 0)$. In this case Q itself is empty, since no transition has an empty pre-set, a fact which follows from Definition 2. It suffices to set $Q' = \emptyset$.

  $(|M| = k + 1)$. Assume that the thesis holds for $|M| \leq k$ and consider the case $|M| = k+1$. Let $t \in M$ be any transition. Since there is no folding pair, for every $t' \in Q - \{t\}$ there must be an index $i_{t'}$ such that $t \not< [{}^\bullet t' \cdot \underline{t'}]_{i_{t'}}$ and $[{}^\bullet t \cdot \underline{t}]_{i_{t'}} \neq [{}^\bullet t' \cdot \underline{t'}]_{i_{t'}}$. Observe that, in particular, $i_{t'} \in I(Q)$. We distinguish two cases:
  · The set $\{([{}^\bullet t' \cdot \underline{t'}]_{i_{t'}}, i_{t'}) \mid t' \in Q - \{t\}\}$ is finite, i.e., it has the form $\{(s_1, i_1), \ldots, (s_l, i_l)\}$. We can thus define $Q_j = \{t' \in Q \mid [{}^\bullet t' \cdot \underline{t'}]_{i_{t'}} =$

$s_j \wedge i_{t'} = i_j\}$ and it holds that $Q = \{t\} \cup Q_1 \cup \ldots \cup Q_l$.

For each of the Q_i we have $|I(Q_i)| < n$ and, by the (outer) induction hypothesis, we can—one after the other—remove almost all of the transitions of $Q_1, \ldots, Q_l$ from N obtaining a net which is still infinite. Hence there is a set $Q' \subseteq Q - \{t\}$ such that $Q - Q'$ is finite and $N - Q'$ is infinite.

Note that by removing Q_i, we might also remove some elements of Q_j with $j > i$. More formally one could resort to an inductive reasoning on l.

· The set $\{([{}^\bullet t' \cdot \underline{t'}]_{i_{t'}}, i_{t'}) \mid t' \in Q - \{t\}\}$ is infinite. First observe that, in this case, the set of places $S' = \{[{}^\bullet t' \cdot \underline{t'}]_{i_{t'}} \mid t' \in Q - \{t\}\}$ is infinite, and, by construction, the places in S' do not causally depend on t. Hence $N' = N - \{t\}$ still contains infinitely many places and therefore infinitely many transitions.

Furthermore the set of minimal elements of $P = Q \cap T_{N'}$ is $M - \{t\}$ which has cardinality k. Thus the (inner) induction hypothesis can be applied to deduce that there is a set $P' \subseteq P$ such that $P - P'$ is finite and $N' - P'$ is infinite.

Finally, if we set $Q' = P' \cup \{t\} \cup \{t' \in Q \mid t <_N t'\}$, clearly $Q' \subseteq Q$. Furthermore $Q - Q' = (Q \cap T_{N'}) - P' = P - P'$ is finite and it holds that $N - Q' = N' - P'$ which is infinite, as desired. □

Proposition 36 (termination). *The algorithm computing the k-covering (see Definition 26) terminates for every graph grammar $\mathcal{G}$ and every $k \in \mathbb{N}$.*

PROOF. Let $\mathcal{G} = (\mathcal{R}, G_\mathcal{R})$ be the input grammar, where $\mathcal{R}$ is a set of rewrite rules and $G_\mathcal{R}$ the start graph. The algorithm produces a sequence $P_0 = (G_0, N_0), P_1 = (G_1, N_1), \ldots$ of Petri graphs. Our aim is to show that this sequence will eventually terminate.

- In parallel to the construction of the covering $\mathcal{C}^k(\mathcal{G})$ we define a sequence of tuples $(N'_0, \beta_0), (N'_1, \beta_1), \ldots$. All the N'_i are irredundant Petri nets satisfying the conditions of Lemma 34 and the $\beta_i : N'_i \to N_i$ are net morphisms (see Fig. A.1 for a graphical representation of this situation).

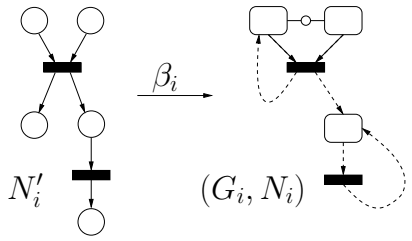


Fig. A.1. Schematic representation of the morphisms β_i.

This sequence is constructed in the following way:

Start tuple: $N'_0 = N_0$, $\beta_0 : N'_0 \to N_0$ is the identity.

Unfolding step: Assume that P_{i+1} is obtained from $P_i = (G_i, N_i)$ by an unfolding step. Let t be the transition added to N_i, with $p_{N_{i+1}}(t) = (L \hookleftarrow K \hookrightarrow R) = r$, and let $\psi_{i+1} : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ be the PG_ι morphism as in Definition 26.

Let $|{}^\bullet t| = k$, $|\underline{t}| = m$ and $|t^\bullet| = l$. For every index j with $1 \le j \le k+m$ take a place s'_j in N'_i such that $\beta_i(s'_j) = [{}^\bullet t \cdot \underline{t}]_j$ (as we will show later, all the β_i are surjective and thus such an s'_j always exists).

Furthermore let $\hat{t}$ be a new transition, not in N'_i, and let $\hat{s}_1, \ldots, \hat{s}_l$ be new places. We construct N'_{i+1} as follows.[7]

$$\begin{aligned} N'_{i+1} = (&S_{N'_i} \cup \{\hat{s}_1, \ldots, \hat{s}_l\}, T_{N'_i} \cup \{\hat{t}\}, \\ &{}^\bullet(\ ) \cup \{\hat{t} \mapsto s'_1 \ldots s'_k\}, (\ )^\bullet \cup \{\hat{t} \mapsto \hat{s}_1 \ldots \hat{s}_l\}, \\ &\underline{(\ )} \cup \{\hat{t} \mapsto s'_{k+1} \ldots s'_{k+m}\}, p_{N'_i} \cup \{\hat{t} \mapsto r\}). \end{aligned}$$

Moreover β_{i+1} is set to

$$\beta_{i+1} = ((\psi_{i+1})_N \circ \beta_i) \cup \{\hat{s}_j \mapsto [t^\bullet]_j \mid 1 \le j \le l\} \cup \{\hat{t} \mapsto t\}$$

where $(\psi_{i+1})_N$ denotes the net component part of the PG_ι morphism ψ_{i+1}.

Folding step: Assume that P_{i+1} is obtained from P_i by a folding step. Let $\psi_{i+1} : (P_i, u_i) \to (P_{i+1}, u_{i+1})$ be a PG_ι morphism as in Definition 22. Set $\beta_{i+1} = (\psi_{i+1})_N \circ \beta_i$.

Note that the described procedure is non-deterministic since the places s'_j in the unfolding step are not uniquely determined.

- By induction on i we can easily show that the following invariants hold:

(i) every net N'_i satisfies the conditions of Lemma 34;

(ii) the mappings β_i are surjective;

(iii) the β_i are pre-net morphisms, i.e., for every transition $t' \in T_{N'_i}$, ${\beta_i}^*({}^\bullet t') = {}^\bullet(\beta_i(t'))$, ${\beta_i}^*(\underline{t'}) = \underline{(\beta_i(t'))}$, and ${\beta_i}^*(t'^\bullet) = (\beta_i(t'))^\bullet$, and, furthermore, $p_{N'_i} = p_{N_i} \circ \beta_i$.

(From this we deduce that $x <_{N'_i} y$ for $x, y \in S_{N'_i} \cup T_{N'_i}$ implies $\beta_i(x) <_{N_i} \beta_i(y)$.)

Moreover, from the fact that the β_i are net morphisms, we can also show, by contradiction, that every N'_i is irredundant. Assume that a new transition $\hat{t}$ with $t = \beta_i(\hat{t})$ is added to N'_i in an unfolding step and suppose that there is already a transition $\hat{u} \in T_{N'_i}$ with the same label such that ${}^\bullet\hat{u} = {}^\bullet\hat{t}$ and $\underline{\hat{u}} = \underline{\hat{t}}$. Considering $u = \beta_i(\hat{u})$, we have ${}^\bullet u = {\beta_i}^*({}^\bullet\hat{u}) = {\beta_i}^*({}^\bullet\hat{t}) = {}^\bullet t$, $\underline{u} = {\beta_i}^*(\underline{\hat{u}}) = {\beta_i}^*(\underline{\hat{t}}) = \underline{t}$ and $p_{N_i}(u) = p_{N'_i}(\hat{u}) = r$. This implies that Condition (U2) of the unfolding step was violated, leading to a contradiction.

[7] For a function $f\colon A \to B$ and $a \notin A$ we denote by $\bar{f} = f \cup \{a \mapsto b\}$ its natural extension with $\bar{f}(a) = b$ and $\bar{f}(a') = f(a')$ for $a' \in A$.

- Now assume that the algorithm does not terminate. This implies that infinitely many unfolding steps are performed (folding steps decrease the size of the graph G_i). Since unfolding steps increase the size of N'_i, while folding steps do not alter its size, it follows that $N' = \bigcup_{i=1}^{\infty} N'_i$ (where union is defined in the obvious way) is infinite. It is easy to check that, since any N'_i satisfies the conditions of Lemma 34 then also the infinite net N' does. In particular, any transition has a finite set of causes and $<$ is acyclic, since adding a new transition $\hat{t}$ and new places $\hat{s}_1, \ldots, \hat{s}_l$ in the unfolding step does not alter the causes of already existing transitions and places. Also, every place has at most one transition as a direct cause. Moreover we never introduce places with no causes, and therefore the size of $Min(N'_i)$ is finite and constant.

  Let $((P, \iota), \{\xi_i : P_i \to P\}_i)$ be the colimit of the chain $((P_i, \iota_i), \psi_i)_i$, which exists by Proposition 16. Since all ψ_i are k-monomorphisms, we can show that the ξ_i are k-monomorphisms as well.

  Let $P = (N, G)$ and for any transition t' in N', such that $p_{N'}(t') = L \hookleftarrow K \hookrightarrow R$, define a function $d'(t') : V_L \cup E_L \to V_G \cup E_G \cup \{k\}$ as follows:

$$d'(t')(x) = \begin{cases} \mu(\xi_i(\beta_i(t')))(x) & \text{if } \mathit{depth}(\xi_i(\beta_i(t')))(x)) < k, \\ k & \text{otherwise} \end{cases}$$

  where i is an index such that t' occurs in N'_i and $\mu(\xi_i(\beta_i(t'))) : L \cup R \to G$ denotes the function associated to transition $\xi_i(\beta_i(t'))$ in the Petri graph P (see Definition 11). Since $\xi_{i+1}(\beta_i(t')) = \xi_i(\beta_i(t'))$, which follows from the properties of the colimit and the definition of the β_i, we can infer that the mapping $d'(t')$ is well-defined for every transition t' in N'. Intuitively, $d'(t')$ records the items of depth smaller than k which are used by the rewriting rule application associated to the transition.

  Then we relabel N', by taking as label for each transition t' the pair $\langle p_N(t'), d'(t') \rangle$. It is easy to realize that, by construction, there are are only finitely many rules in the grammar and finitely many nodes and edges of depth smaller than k in P, and thus the set of such new labels is finite.

  Therefore we can apply Lemma 35 and obtain the existence of a folding pair $\hat{u}, \hat{t} \in T_{N'}$ where $\hat{u} \neq \hat{t}$, both transitions have the same label and for all j with $1 \leq j \leq |\, {}^\bullet\hat{u} \cdot \underline{\hat{u}}| = |\, {}^\bullet\hat{t} \cdot \underline{\hat{t}}|$ it holds that either $[{}^\bullet\hat{u} \cdot \underline{\hat{u}}]_j = [{}^\bullet\hat{t} \cdot \underline{\hat{t}}]_j$ or $\hat{u} < [{}^\bullet\hat{t} \cdot \underline{\hat{t}}]_j$.
- Assume that $\hat{t}$ was added when N'_{i+1} was constructed from N'_i, by means of a step which is (necessarily) an unfolding step, adding the transition $\beta_{i+1}(\hat{t}) = t$ to N_i. We will show that this unfolding step could never have been applied, thus obtaining a contradiction.

  Since the causes of an already existing transition are never altered during the construction of the nets N'_i, the folding pair $\hat{u}, \hat{t}$ is already present in N'_{i+1}. Let $u = \beta_{i+1}(\hat{u})$. Since $\hat{u}$ appears in N'_i, it holds that $u = \beta_{i+1}(\hat{u}) = \psi_{i+1}(\beta_i(\hat{u})) \neq t$, since unfolding steps do not merge any transitions. Since the

labelling p of the pre-nets is preserved by β_{i+1}, it also holds that $p_{N_{i+1}}(u) = p_{N_{i+1}}(t) = r = (L \hookleftarrow K \hookrightarrow R)$.

We first show that u, t is a folding pair: for every index j it holds that $\beta_{i+1}([^\bullet\hat{u} \cdot \underline{\hat{u}}]_j) = [^\bullet u \cdot \underline{u}]_j$ and $\beta_{i+1}([^\bullet\hat{t} \cdot \underline{\hat{t}}]_j) = [^\bullet t \cdot \underline{t}]_j$. As mentioned above it either holds that $[^\bullet\hat{u} \cdot \underline{\hat{u}}]_j = [^\bullet\hat{t} \cdot \underline{\hat{t}}]_j$ which implies $[^\bullet u \cdot \underline{u}]_j = [^\bullet t \cdot \underline{t}]_j$, or $\hat{u} < [^\bullet\hat{t} \cdot \underline{\hat{t}}]_j$ which implies $u < [^\bullet t \cdot \underline{t}]_j$ (immediate by definition of pre-net morphisms).

- Let $\varphi_u = \mu_{i+1}(u)_{|L}$ and $\varphi_t = \mu_{i+1}(t)_{|L}$. Consider any node or edge $x \in V_L \cup E_L$. Since the mappings d' are part of the transition labels, we have $d'(\hat{u}) = d'(\hat{t})$. Since ξ_{i+1} is a Petri graph morphism and the μ-components are unique, it holds that $\xi_{i+1} \circ \mu_{i+1}(t) = \mu(\xi_{i+1}(t))$ for every $t \in T_{N_{i+1}}$. Hence for every x in $V_L \cup E_L$ either $\xi_{i+1}(\varphi_u(x))) = \xi_{i+1}(\varphi_t(x))$ or $\mathit{depth}(\xi_{i+1}(\varphi_u(x))) \geq k$ and $\mathit{depth}(\xi_{i+1}(\varphi_t(x))) \geq k$. Thus, since ξ_{i+1} is a k-monomorphism, either $\varphi_u(x) = \varphi_t(x)$ or $\mathit{depth}(\varphi_u(x)) \geq k$ and $\mathit{depth}(\varphi_t(x)) \geq k$.

  Since the causes of $\varphi_t(E_L)$ do not change during the unfolding step, the condition for the application of the folding step is satisfied, which forbids the application of the unfolding step, leading to a contradiction (see Condition (U3) in Definition 26). □

Proposition 39 (local confluence). *Let (P, u) $\dashrightarrow$ (P_i, u_i) for $i \in \{1, 2\}$. Then there is a Petri graph (P', u') such that (P_i, u_i) $\dashrightarrow^*$ (P', u') for $i \in \{1, 2\}$.*

PROOF. The proof mainly relies on the fact that both folding and unfolding operations can be described as colimits in a suitable category of Petri graphs. Then a general categorical result that ensures the commutativity of colimits can be exploited. Finally, things must be accommodated to take into account also the applicability conditions of folding and unfolding steps as described in the algorithm (see Definition 26).

Confluence without application conditions. If (P, u) $\dashrightarrow$ (P_i, u_i), then there are PG_ι morphisms $\varphi_i : (P, u) \to (P_i, u_i)$. Both folding and unfolding steps are expressed in terms of colimits. Thus, since colimits "commute", it holds that:

- (P, u) $\dashrightarrow^{unf}_{r_i, \psi_i}$ (P_i, u_i), $i \in \{1, 2\}$ implies $(P_1, u_1) \rightsquigarrow^{unf}_{r_2, \varphi_1 \circ \psi_2}$ $(P', \varphi'_1(u_1))$ and $(P_2, u_2) \rightsquigarrow^{unf}_{r_1, \varphi_2 \circ \psi_1}$ $(P', \varphi'_2(u_2))$ for some Petri graph P'.
- (P, u) $\dashrightarrow^{fold}_{r_i, \psi_i, \eta_i}$ (P_i, u_i), $i \in \{1, 2\}$ implies $(P_1, u_1) \rightsquigarrow^{fold}_{r_2, \varphi_1 \circ \psi_2, \varphi_1 \circ \eta_2}$ (P', u') and $(P_2, u_2) \rightsquigarrow^{fold}_{r_1, \varphi_2 \circ \psi_1, \varphi_2 \circ \eta_1}$ (P', u') for some Petri graph (P', u').
- (P, u) $\dashrightarrow^{fold}_{r_1, \psi_1, \eta_1}$ (P_1, u_1) and (P, u) $\dashrightarrow^{unf}_{r_2, \psi_2}$ (P_2, u_2) implies $(P_1, u_1) \rightsquigarrow^{unf}_{r_2, \varphi_1 \circ \psi_2}$ (P', u') and $(P_2, u_2) \rightsquigarrow^{fold}_{r_1, \varphi_2 \circ \psi_1, \varphi_2 \circ \eta_1}$ (P', u') for some Petri

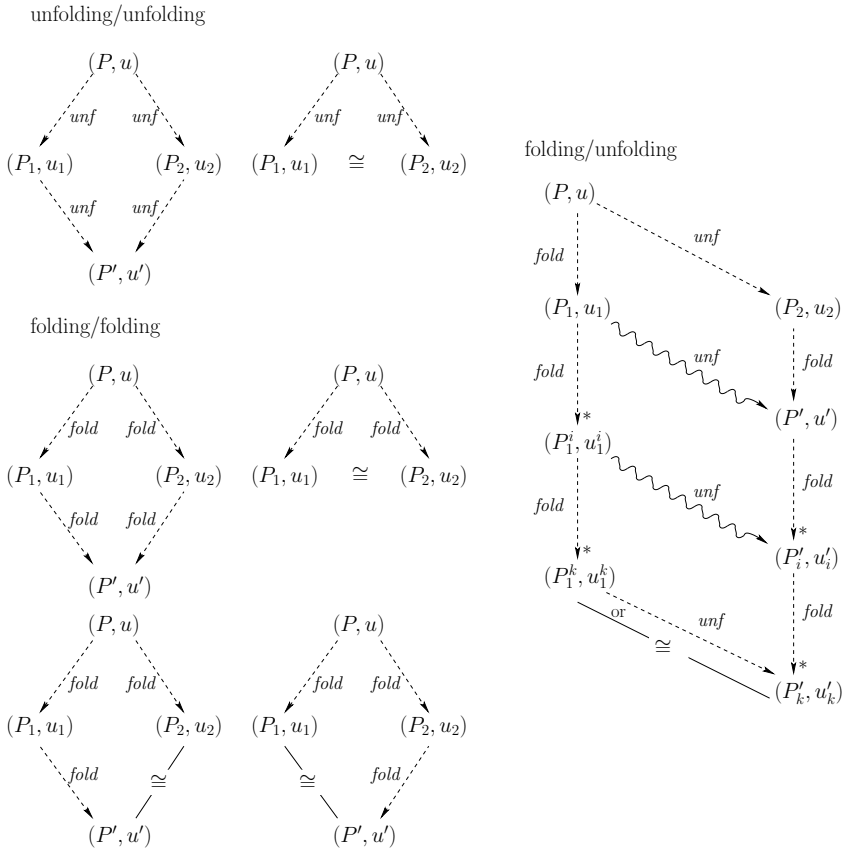


Fig. A.2. Confluence of the rewriting system.

graph (P', u').

Confluence with application conditions. The above considerations show a confluence result, but in a setting where the application conditions are not considered. Next we show that the rewriting system is still confluent if we take into account such conditions. To this aim we prove that, when the algorithm can perform two diverging steps, it is always possible to perform other steps in order to produce a common Petri graph. We distinguish several cases according to the kind of diverging steps:

unfolding/unfolding: Let $(P,u) \dashrightarrow^{unf}_{r_i,\psi_i} (P_i,u_i)$, $i \in \{1,2\}$ where $P = (G,N)$, $P_i = (G_i,N_i)$, $r_i = (L_i \hookleftarrow K_i \hookrightarrow R_i)$ and $\psi_i : L_i \to G$.

We consider Conditions (U1)–(U3) for the unfolding of rule r_2 at the match $\varphi_1 \circ \psi_2 : L_2 \to G_1$, showing that either they are satisfied, thus allowing to reach a common Petri graph, or $(P_1,u_1) \cong (P_2,u_2)$ (see Fig. A.2). (The same argument applies to the unfolding of the match $\varphi_2 \circ \psi_1$ of rule r_1 in P_2.)

(U1) The state $(\varphi_1 \circ \psi_2)^{\oplus}(E_{L_2})$ is coverable. This follows, by Lemma 37, from the fact that ${\psi_2}^{\oplus}(E_{L_2})$ is coverable.

(U2) If this condition is not satisfied, i.e., if there is a transition $t \in$

T_{N_1} such that $p_{N_1}(t) = r_2$, ${}^\bullet t = (\varphi_1 \circ \psi_2)^*(\lambda(E_{L_2} - E_{K_2}))$ and $\underline{t} = (\varphi_1 \circ \psi_2)^*(\lambda(E_{K_2}))$, then, necessarily, t has been introduced by the last unfolding step, since otherwise the unfolding of r_2 would not have been possible in P.

This implies that $p_{N_1}(t) = r_1 = r_2$ and $\varphi_1 \circ \psi_1 = \varphi_1 \circ \psi_2$, since left-hand sides do not contain isolated nodes and thus a match is determined uniquely by the images of the edges. Since φ_1 is injective it follows that $\psi_1 = \psi_2$. Therefore both steps unfold the same match of a left-hand side and thus the resulting Petri graphs P_1, P_2 are isomorphic.

(U3) Since the unfolding step does not change the causes of $(\varphi_1 \circ \psi_2)^*(E_{L_2} - E_{K_2})$ and $(\varphi_1 \circ \psi_2)^*(\lambda(E_{K_2}))$, and depth is not changed by unfolding steps, there cannot be other matches ψ_2' of L_2 such that the folding condition holds for the pair ψ_2' and $\varphi_1 \circ \psi_2$.

folding/folding: Let $(P, u) \dashrightarrow^{fold}_{r_i,\psi_i,\eta_i} (P_i, u_i)$, $i \in \{1, 2\}$ where $P = (G, N)$, $P_i = (G_i, N_i)$, $r_i = (L_i \hookleftarrow K_i \hookrightarrow R_i)$ and $\psi_i, \eta_i : L_i \to G$. We show that either the application conditions for the folding of the two occurrences $\varphi_1 \circ \psi_2, \varphi_1 \circ \eta_2 : L_2 \to G_1$ are satisfied or that $(P_1, u_1) \cong (P', u')$ (see Fig. A.2). (The same argument applies for the corresponding folding in P_2.)

Assume that $(P_1, u_1) \ncong (P', u')$. Then:

(F1) The matches $\varphi_1 \circ \psi_2, \varphi_1 \circ \eta_2 : L_2 \to G_1$ are distinct. In fact, if they were equal, since by the observation in the first part $(P_1, u_1) \rightsquigarrow_{r_2,\varphi_1\circ\psi_2,\varphi_1\circ\eta_2} (P', u')$, using Lemma 38 we would conclude that $(P_1, u_1) \cong (P', u')$.

(F2) The match $(\varphi_1 \circ \eta_2)^{\oplus}(E_{L_2})$ is coverable. This follows, by Lemma 37, from the fact that ${\eta_2}^{\oplus}(E_{L_2})$ is coverable.

(F3) By hypothesis, the folding condition for the occurrences $\psi_2, \eta_2 : L_2 \to G$ is satisfied, i.e., there is a transition t with $p_N(t) = r_2$ and ${\psi_2}^*(\lambda(E_{L_2} - E_{K_2})) = {}^\bullet t$, ${\psi_2}^*(\lambda(E_{K_2})) = \underline{t}$ and for every $e \in E_L$ it either holds that $\psi_2(e) = \eta_2(e)$ or $t < \eta_2(e)$. Since φ_1 is a PG_ι morphism and morphisms preserve causality, the folding condition (F3) is satisfied also for $\varphi_1 \circ \psi_2$ and $\varphi_1 \circ \eta_2$, as witnessed by transition $t' = \varphi_1(t)$.

(F4) Since φ_1 is a PG_ι morphism, by Lemma 27, it never decreases depth. Hence the depth condition for the new matches is surely satisfied.

folding/unfolding: Let $(P, u) \dashrightarrow^{fold}_{r_1,\psi_1,\eta_1} (P_1, u_1)$ and $(P, u) \dashrightarrow^{unf}_{r_2,\psi_2} (P_2, u_2) = (G_2, N_2)$ where $P = (G, N)$ and $P_i = (G_i, N_i)$. This is the most difficult case, since the application of the folding step might invalidate Condition (U3) of the unfolding step.

(F1)–(F4), (U1), (U2) With the same argument as above (case folding/folding), we can show either that the matches $\varphi_2 \circ \psi_1$, $\varphi_2 \circ \eta_1$ in (P_2, u_2) satisfy the folding condition (Conditions (F1)–(F4)) and therefore $(P_2, u_2) \dashrightarrow^{fold}_{r_1,\varphi_2\circ\psi_1,\varphi_2\circ\eta_1} (P', u')$, or $(P_2, u_2) \cong (P', u')$.

Furthermore it immediately follows that Condition (U1) (coverability) is also satisfied for the match $\varphi_1 \circ \psi_2$. Finally Condition (U2) is either satisfied or, by Lemma 38, $(P_2, u_2) \cong (P', u')$.

(U3) In order to close the diamond, it is necessary to deal with Condition (U3). This is done according to the steps below (see Fig. A.2):

- In P_1 the morphism $\varphi_1 \circ \psi_2 : L_2 \to G_1$ might be part of a folding pair $\zeta_1, \varphi_1 \circ \psi_2 : L_2 \to G_1$, which forbids unfolding, and after folding this pair, the image of L_2 might again be part of a folding pair, etc. More precisely, there is a (possibly empty) sequence of transitions of the form

$$(P_1, u_1) = (P_1^0, u_1^0) \dashrightarrow^{fold}_{r_2,\zeta_1,\varphi_1\circ\psi_2} (P_1^1, u_1^1) \dashrightarrow^{fold}_{r_2,\zeta_2,\varphi^1\circ\varphi_1\circ\psi_2} \ldots \dashrightarrow^{fold}_{r_2,\zeta_k,\varphi^k\circ\ldots\circ\varphi^1\circ\varphi_1\circ\psi_2} (P_1^k, u_1^k)$$

  where the $\varphi^{i+1} : (P_1^i, u_1^i) \to (P_1^{i+1}, u_1^{i+1})$ are PG_ι morphisms. We assume that no further folding steps of this form are applicable in the Petri graph P_1^k. Observe that such a finite sequence exists since a folding step decreases the size of a Petri graph.
- Now we can construct another sequence of Petri graphs starting from (P', u') of the form $(P', u') = (P'_0, u'_0), \ldots, (P'_k, u'_k)$ such that (P'_i, u'_i) and (P'_{i+1}, u'_{i+1}) are either isomorphic, or there is a folding step from one to the other, and furthermore

$$(P_1^i, u_1^i) \rightsquigarrow^{unf}_{r_2,\varphi^i\circ\ldots\circ\varphi^1\circ\varphi_1\circ\psi_2} (P'_i, u'_i) \tag{A.1}$$

  This is shown by induction on i:

  ($i = 0$) Trivial.

  ($i \to i+1$) Assume that (A.1) holds and let $\varphi'_i : (P_1^i, u_1^i) \to (P'_i, u'_i)$ be the corresponding PG_ι morphism. As in the case (folding/folding) we can argue that the morphisms $\varphi'_i \circ \zeta_i$ and $\varphi'_i \circ \varphi^i \circ \ldots \circ \varphi^1 \circ \varphi_1 \circ \psi_2$ are either equal or satisfy the conditions for the application of a folding rule. In the former case we set $(P'_{i+1}, u'_{i+1}) = (P'_i, u'_i)$ and in the latter case we define $(P'_{i+1}, u'_{i+1}) = \mathsf{fold}((P'_i, u'_i), r_2, \varphi'_i \circ \zeta_i, \varphi'_i \circ \varphi^i \circ \ldots \circ \varphi^1 \circ \varphi_1 \circ \psi_2)$.

  Since colimits commute it also holds that

$$(P_1^{i+1}, u_1^{i+1}) \rightsquigarrow^{unf}_{r_2,\varphi^{i+1}\circ\ldots\circ\varphi^1\circ\varphi_1\circ\psi_2} (P'_{i+1}, u'_{i+1})$$

- Finally we reach two Petri graphs (P_1^k, u_1^k), with $P_1^k = (G_1^k, N_1^k)$, and (P'_k, u'_k), with $P'_k = (G'_k, N'_k)$. The first one is produced from (P_1, u_1), the second one from (P_2, u_2), by a sequence of folding steps, and condition (A.1) holds for $i = k$.

  Now Condition (U3) is satisfied and the occurrence of L_2 in G_1^k is still coverable since coverability is preserved by application of morphisms (see Lemma 37) and also the depth condition still holds. The only condition that might forbid the application of the unfolding step is the existence of a transition $t \in T_{N_1^k}$ such that ${}^\bullet t = (\varphi^k \circ \ldots \circ \varphi^1 \circ \varphi_1 \circ \psi_2)^* \lambda(E_{L_2} - E_{K_2})$ and $\underline{t} = (\varphi^k \circ \ldots \circ \varphi^1 \circ \varphi_1 \circ \psi_2)^* \lambda(E_{K_2})$. But in this case, by Lemma 38, we

deduce that $(P_1^k, u_1^k) \cong (P'_k, u'_k)$. Otherwise, in absence of such a transition, we directly obtain $(P_1^k, u_1^k) \dashrightarrow^{unf}_{r_2, \varphi^k \circ \ldots \circ \varphi^1 \circ \varphi_1 \circ \psi_2} (P'_k, u'_k)$.

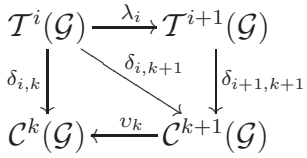
Observe, in particular, that all conditions, apart from Condition (U3), allow us to close the diamond in at most one step. □

Lemma 41. *If Condition (U3) is violated a finite number of times during the construction of the covering, the algorithm of Definition 26 still terminates with the same unique result.*

PROOF. (Sketch) The proof can be carried out by slightly changing the proofs of the previous results:

- The algorithm still *terminates*, even if we violate Condition (U3) a finite number of times. This result can be obtained by a slight modification of the proof of Proposition 36. Attach a special label to all transitions which have been created in violation of Condition (U3), in order to distinguish them from all other transitions. The set of labels remains finite and thus the termination proof can be carried out as before. Note that none of the two transitions belonging to the detected folding pair will be marked by such a special label.
- The folding and unfolding steps of Definition 26 are globally confluent, even if we do not check Condition (U3) (although, however, the algorithm might not terminate without (U3)). This follows from the fact that every diamond can be completed in at most one step (see the remark at the end of the proof of Proposition 39). Then one can show global confluence by simple induction on the number of steps.
- Finally observe that the result of the algorithm is unique, even if Condition (U3) is violated a finite number of times. The argument goes as follows: Let $[G_0]$ be the initial Petri graph, let $\mathcal{U}$ be the result of the standard approximated unfolding procedure and let $\mathcal{U}'$ be the Petri graph one obtains by violating Condition (U3) finitely often. Global confluence implies that there is a Petri graph $\mathcal{U}''$ which is reachable from both $\mathcal{U}$ and $\mathcal{U}'$. But since no folding/unfolding steps are possible in $\mathcal{U}$ and $\mathcal{U}'$ we can infer that $\mathcal{U}$ and $\mathcal{U}'$ are isomorphic. □

Proposition 42 (k-Monomorphisms υ_k and $\delta_{i,k}$). *For any $k, i \in \mathbb{N}$ there are k-monomorphisms $\upsilon_k \colon \mathcal{C}^{k+1}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and $\delta_{i,k} \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ such that the following diagram commutes. The morphisms υ_k and $\delta_{i,k}$ for $i \geq k$ are k-isomorphisms. The morphisms $\lambda_i \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{T}^{i+1}(\mathcal{G})$ are those forming the truncation tower, as introduced in Definition 22.*



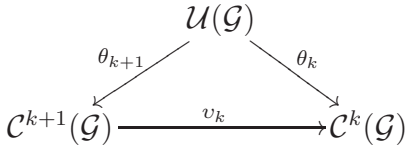
PROOF. To obtain morphism $\upsilon_k \colon \mathcal{C}^{k+1}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ proceed as follows: Construct $\mathcal{C}^{k+1}(\mathcal{G})$ according to the algorithm in Definition 26. Now, the sequence of folding and unfolding steps can be considered as part of the algorithm for the construction of $\mathcal{C}^k(\mathcal{G})$, where Condition (U3) might have been violated a certain number of times. Hence, one can continue folding and unfolding, now respecting Condition (U3), and Lemma 41 implies that we terminate with a unique Petri graph, which must be isomorphic to $\mathcal{C}^k(\mathcal{G})$. Since any step in the algorithm transforms a Petri graph P_i into a Petri graph P_{i+1} and there is a PG_ι morphism $\psi_i \colon (P_i, u_i) \to (P_{i+1}, u_{i+1})$, composing such morphisms we can obtain the desired $\upsilon_k \colon \mathcal{C}^{k+1}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$.

Furthermore according to Lemma 31, the morphisms ψ_i are k-monomorphism. Since no new item of depth smaller than k is created, they are in fact k-isomorphisms as well. Hence, by Lemma 30, also their composition, υ_k, is a k-isomorphism.

By a similar argument one can obtain the morphisms $\delta_{i,k} \colon \mathcal{T}^i(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$, which are k-monomorphisms for $k > i$ and k-isomorphisms for $k \leq i$.

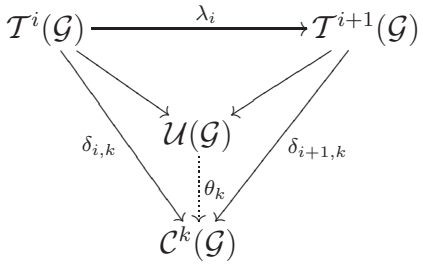
It can be shown, by induction on i that for any $i, j \in \mathbb{N}$, $\delta_{i,j} \colon \mathcal{T}^i(\mathcal{R}, G_\mathcal{R}) \to \mathcal{C}^j(\mathcal{R}, G_\mathcal{R})$ is the unique morphism between the two marked Petri graphs (the image of the start graph is fixed and the rest follows by exploiting irredundancy). Hence the commutativity of the diagram immediately follows. □

Proposition 43. *For every index k there is a marked* PG *mor¡phism which is a k-isomorphism $\theta_k \colon \mathcal{U}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ and such that the following diagram commutes.*



PROOF. Since the triangle, consisting of the Petri graphs $\mathcal{T}^i(\mathcal{G})$, $\mathcal{T}^{i+1}(\mathcal{G})$ and $\mathcal{C}^k(\mathcal{G})$ in the diagram in Proposition 42 commutes and the full unfolding

$\mathcal{U}(\mathcal{G})$ is the colimit of the truncation tower, consisting of the morphisms λ_i (see Definition 23), it follows that there are mediating morphisms $\theta_k : \mathcal{U}(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ such that the following diagram commutes:



It is easy to show that every morphism $\mathcal{T}^k(\mathcal{G}) \to \mathcal{U}(\mathcal{G})$ satisfies Condition (iii) of Definition 28. Since the morphism $\delta_{k,k}$ is a k-isomorphism, we can apply Lemma 30 and conclude that the morphisms θ_k are k-isomorphisms.

Since we can show again that θ_k is the unique PG_ι morphism from $\mathcal{U}(\mathcal{G})$ to $\mathcal{C}^k(\mathcal{G})$ (the image of the initial marking is fixed and the rest follows by exploiting irredundancy), we obtain $\upsilon_k \circ \theta_k = \theta_{k+1}$. $\square$

Proposition 46 (unfolding as limit of the coverings). *The limit in the category PG_ι of the covering tower $\mathcal{C}^0(\mathcal{G}) \overset{\upsilon_0}{\leftarrow} \ldots \mathcal{C}^k(\mathcal{G}) \overset{\upsilon_k}{\leftarrow} \mathcal{C}^{k+1}(\mathcal{G}) \overset{\upsilon_{k+1}}{\leftarrow} \ldots$ is the full unfolding $\mathcal{U}(\mathcal{G})$ of the graph grammar.*

PROOF. We show that the covering tower

$$\mathcal{C}^0(\mathcal{G}) \overset{\upsilon_0}{\leftarrow} \ldots \mathcal{C}^k(\mathcal{G}) \overset{\upsilon_k}{\leftarrow} \mathcal{C}^{k+1}(\mathcal{G}) \overset{\upsilon_{k+1}}{\leftarrow} \ldots$$

has a limit

$$((A, u_A), \pi_k : (A, u_A) \to \mathcal{C}^k(\mathcal{G}))$$

in the category PG_ι which is defined as follows. The Petri graph A with initial marking u_A is a subset of the (componentwise) product $\Pi_{k\in\mathbb{N}} \mathcal{C}^k(\mathcal{G})$: Given an item (node, edge, transition) $x \in \Pi_{k\in\mathbb{N}} \mathcal{C}^k(\mathcal{G})$

$$x = \langle x_k \rangle_k \in A \quad \text{iff} \quad \begin{array}{l} \text{(i) } \forall k.\ x_k = \upsilon_k(x_{k+1}) \\ \text{(ii) } \exists h.\ \forall k > h.\ \mathit{depth}(x_k) = h \end{array}$$

The connection function, the pre- and post-set function and the context are defined componentwise. E.g., if $x = \langle x_k \rangle_k \in A$ is an edge, then the i-th node connected to the edge is

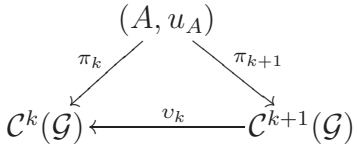
$$[c_G(x)]_i = \langle [c_{G_k}(x_k)]_i \rangle_k$$

where G_k is the graph underlying $\mathcal{C}^k(\mathcal{G})$. The initial marking u_A is a sequence with $|u_0|$ elements such that $[u_A]_i = \langle [u_k]_i \rangle_k$ where u_i is the initial marking of $\mathcal{C}^k(\mathcal{G})$.

A long but straightforward calculation allows to prove that A is a well-defined Petri graph. Let us verify, for instance, that when x is an edge in the graph G underlying A then $[c_G(x)]_i = \langle [c_{G_k}(x_k)]_i \rangle_k$ is actually a node in G, i.e., there exists some $h' \in \mathbb{N}$ such that $\mathit{depth}([c_{G_k}(x_k)]_i) = h'$ for any $k > h'$. By hypothesis, since x is an edge in A, there exists h such that $\mathit{depth}(x_k) = h$ for any $k > h$. Now, it is easy to show that if for an edge e in $\mathcal{C}^k(\mathcal{R}, G_{\mathcal{R}})$ it holds $\mathit{depth}(e) = h < k$, then $\mathit{depth}(v) \leq h$ for all its attached nodes v. Therefore we can deduce that $\mathit{depth}([c_{G_{h+1}}(x_{h+1})]_i) = h' < h+1$. Therefore, since morphisms υ_k are k-isomorphisms (see Proposition 42) we deduce $\mathit{depth}([c_{G_k}(x_k)]_i) = h'$ for any $k > h'$, as desired.

Let us show that (A, u_A) is an object in PG_ι. Let $t = \langle t_k \rangle_k$ be a transition in A. To show that t can be fired, first observe that, according to Condition (ii) above, there exists an index h such that for all $k > h$, $\mathit{depth}(t_k) = h$. Since the $(h+1)$-covering is an element of PG_ι, it holds that t_{h+1} can be fired, i.e., there is a firing sequence $t^1_{h+1}, \ldots, t^m_{h+1} = t_{h+1}$ in $\mathcal{C}^{h+1}(\mathcal{G})$. Since all υ_k for $k > h$ are h-isomorphisms, for each $k > h$ there are unique transitions $t^1_k, \ldots, t^m_k$ such that $(\upsilon_{h+1} \circ \cdots \circ \upsilon_{k-1})(t^j_k) = t^j_{h+1}$. Moreover, since each υ_k preserves the initial marking and it is a k-isomorphism, every sequence $t^1_k, \ldots, t^m_k$ is a firing sequence. The same holds for the sequence $t^1_\ell, \ldots, t^m_\ell$ with $\ell \leq h$ defined by $t^j_\ell = (\upsilon_\ell \circ \cdots \circ \upsilon_h)(t^j_{h+1})$. Hence $\langle t^1_k \rangle_k, \ldots, \langle t^m_k \rangle_k$ is a firing sequence in A and $\langle t^m_k \rangle_k = t$. A similar argument shows that every edge in A can be covered.

The projections $\pi_k : (A, u_A) \to \mathcal{C}^k(\mathcal{G})$ from the limit into the coverings are defined in the obvious way, i.e., for any $x \in A$ we define $\pi_k(x) = x_k$. A tedious but trivial calculation allows to prove that each π_k is a well-defined marked Petri graph morphism. Commutativity of the diagrams



immediately follows by construction.

To verify that $((A, u_A), \pi_k : (A, u_A) \to \mathcal{C}^k(\mathcal{G}))$ is the limit of the covering tower, let us consider another cone $((A', u_{A'}), \pi'_k : (A', u_{A'}) \to \mathcal{C}^k(\mathcal{G}))$. The unique mediating Petri graph morphism $\varphi : (A', u_{A'}) \to (A, u_A)$ is defined, on any item y in A', as

$$\varphi(y) = \langle \pi'_k(y) \rangle_k$$

Observe that φ is a well-defined Petri graph morphism. The only delicate point concerns the proof of the fact that

$$\varphi(y) = \langle \pi'_k(y) \rangle_k$$

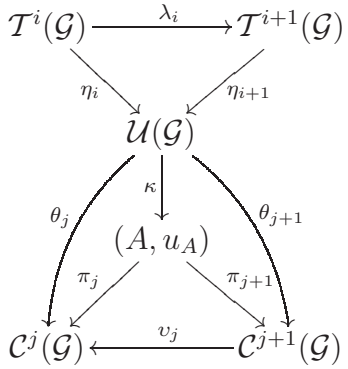
is an element of A. Clearly, by construction for any k we have $\upsilon_k(\pi'_{k+1}(y)) = \pi'_k(y)$. Hence it only remains to verify that there exists a h' such that $\forall k > h'$. $depth(\pi'_k(y)) = h'$. Fix an element $y \in A'$. Since each item in A' is coverable/firable, we can consider the least number h of steps necessary to cover/fire y. Then also $\pi'_{h+1}(y)$ is coverable/firable in h steps. Hence $depth(\pi'_{h+1}(y)) = h' < h$, since every item reachable in $\mathcal{C}^k(\mathcal{G}))$ in $h < k$ steps has depth smaller than or equal to h. Furthermore, since morphisms υ_k are isomorphisms on items of depth smaller than k, we can conclude that for any $k > h'$ we have $depth(\pi'_k(y)) = h'$, as desired. A similar argument is valid for nodes.

Note that to ensure commutativity, i.e., to guarantee that $\pi_k \circ \varphi = \pi'_k$ we are forced to define φ in this way. Hence uniqueness of the mediating morphism φ follows.

We finally prove that (A, u_A) and $\mathcal{U}(\mathcal{G})$ are isomorphic. We know from Proposition 43 that $(\mathcal{U}(\mathcal{G}), \theta_k)$ is a cone for the covering tower and thus, by limit properties, we obtain a unique mediating morphism $\kappa : \mathcal{U}(\mathcal{G}) \to (A, u_A)$ with $\pi_k \circ \kappa = \theta_k$.

Let us show that κ is an isomorphism:

- κ *is injective*: Let x and y in $\mathcal{U}(\mathcal{G})$ such that $\kappa(x) = \kappa(y)$. By construction of the colimit $\mathcal{U}(\mathcal{G})$ (see Proposition 16), if we denote by $\eta_i : \mathcal{T}^i(\mathcal{G}) \to \mathcal{U}(\mathcal{G})$ the embeddings of each truncation into the colimit, there must be $h, k \in \mathbb{N}$, x' in $\mathcal{T}^h(\mathcal{G})$ and y' in $\mathcal{T}^k(\mathcal{G})$ such that $\eta_h(x') = x$ and $\eta_k(y') = y$. Since for any i, we have $\eta_{i+1} \circ \lambda_i = \eta_i$, we can safely assume that $h = k$. By Propositions 42 and 43, we know that $\delta_{k,k} : \mathcal{T}^k(\mathcal{G}) \to \mathcal{C}^k(\mathcal{G})$ is a k-isomorphism and $\delta_{k,k} = \theta_k \circ \eta_k = \pi_k \circ \kappa \circ \eta_k$ (see the proof of Proposition 43). Thus $\delta_{k,k}(x') = \pi_k(\kappa(\eta_k(x'))) = \pi_k(\kappa(\eta_k(y'))) = \delta_{k,k}(y')$. Hence $x' = y'$ and thus $x = y$.
- κ *is surjective*: Let $(x_h)_h$ in (A, u_A). Hence, by construction, there exists k such that for any $h \geq k$, $depth(x_h) = k$. Since θ_k, as defined in Proposition 43, is a k-isomorphism, it follows that there exists x in $\mathcal{U}(\mathcal{G})$ such that $\theta_k(x) = x_k$. Now it is easy to conclude that $\kappa(x) = (x_h)_h$. In fact, for any index $h \leq k$, notice that $\pi_h(\kappa(x)) = \theta_h(x) = \upsilon_h \circ \ldots \circ \upsilon_{k-1} \circ \theta_k(x) = x_h$. Instead, if $h > k$, we have that $\upsilon_k \circ \ldots \circ \upsilon_{h-1} \circ \theta_h(x) = \theta_k(x) = x_k = \upsilon_k \circ \ldots \circ \upsilon_{h-1}(x_h)$. Since morphisms υ_h are k-isomorphisms for $h \geq k$ we conclude that $x_h = \theta_h(x)$, which is the desired result since $\theta_h(x) = \pi_h(\kappa(x))$.



Therefore the limit (A, u_A) of the covering tower is isomorphic to $\mathcal{U}(\mathcal{G})$. □

References

[1] J. Adamek, H. Herrlich, and G.E. Strecker. *Abstract and Concrete Categories - The Joy of Cats*. Wiley, 1990.

[2] P. Aziz Abdulla, B. Jonsson, M. Kindahl, and D. Peled. A general approach to partial order reductions in symbolic verification. In A.J. Hu and M.Y. Vardi, editors, *Proceedings of CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 379–390. Springer Verlag, 1998.

[3] P. Baldan, A. Corradini, J. Esparza, T. Heindel, B. König, and V. Kozioura. Verifying red-black trees. In *Proceedings of COSMICAH '05*, 2005. Proceedings available as report RR-05-04 (Queen Mary, University of London).

[4] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR '01*, volume 2154 of *Lecture Notes in Computer Science*, pages 381–395. Springer Verlag, 2001.

[5] P. Baldan, A. Corradini, and B. König. Static analysis of distributed systems with mobility specified by graph grammars—a case study. In H. Ehrig, B. Krämer, and A. Ertas, editors, *Proceedings of IDPT '02 (Sixth International Conference on Integrated Design & Process Technology)*. Society for Design and Process Science, 2002.

[6] P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: an unfolding-based approach. In P. Gardner and N. Yoshida, editors, *Proceedings of CONCUR 2004*, volume 3170 of *Lecture Notes in Computer Science*, pages 83–98. Springer Verlag, 2004.

[7] P. Baldan, A. Corradini, B. König, and A. Lluch Lafuente. A temporal graph logic for verification of graph transformation systems. In *Proceedings of WADT'06*, volume 4409 of *Lecture Notes in Computer Science*, pages 1–20. Springer Verlag, 2007.

[8] P. Baldan, A. Corradini, and U. Montanari. Concatenable graph processes: relating processes and derivation traces. In S. Larsen, K. Skyum and G. Winskel, editors, *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 283–295. Springer Verlag, 1998.

[9] P. Baldan, A. Corradini, U. Montanari, and L. Ribeiro. Unfolding Semantics of Graph Transformation. *Information and Computation*, 205:733–782, 2007.

[10] P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proceedings of ICGT'02*, volume 2505 of *Lecture Notes in Computer Science*, pages 14–30. Springer Verlag, 2002.

[11] P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Proceedings of SAS'03 (International Static Analysis Symposium)*, volume 2694 of *Lecture Notes in Computer Science*, pages 255–272. Springer Verlag, 2003.

[12] Paolo Baldan. *Modelling concurrent computations: from contextual Petri nets to graph grammars*. PhD thesis, Department of Computer Science, University of Pisa, 2000. Available as technical report n. TD-1/00.

[13] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial models for Petri nets. *Information and Computation*, 170(2):207–236, 2001.

[14] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer Verlag, 2000.

[15] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

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

[17] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):253–291, 1997.

[18] B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 2002.

[19] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Formal Models and Semantics, Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier Science, 1990.

[20] F.L. Dotti, L. Foss, L. Ribeiro, and O. Marchi Santos. Verification of distributed object-based systems. In E. Najm, U. Nestmann, and P. Stevens, editors, *Proceedings of FMOODS '03*, volume 2884 of *Lecture Notes in Computer Science*, pages 261–275. Springer Verlag, 2003.

[21] F.L. Dotti, B. König, O. Marchi dos Santos, and L. Ribeiro. A case study: Verifying a mutual exclusion protocol with process creation using graph transformation systems. Technical Report 08/2004, Universität Stuttgart, 2004.

[22] H. Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Proceedings of the 1st International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69. Springer Verlag, 1979.

[23] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.3: Concurrency, Parallellism, and Distribution*. World Scientific, 1999.

[24] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[25] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, 1994.

[26] F. Gadducci. Graph rewriting for the π-calculus. *Mathematical Structures in Computer Science*, 17(3):407–437, 2007.

[27] F. Gadducci, R. Heckel, and M. Koch. A fully abstract model for graph-intepreted temporal logic. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *6th International Workshop on Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[28] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In T. Nipkow, editor, *Proceedings of 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer Verlag, 1998.

[29] R. Heckel. Compositional verification of reactive systems specified by graph transformation. In E. Astesiano, editor, *Proceedings of FASE'98*, volume 1382 of *Lecture Notes in Computer Science*, pages 138–153. Springer Verlag, 1998.

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

[31] P. Kelb. *Abstraktionstechniken für automatische Verifikationsmethoden*. PhD thesis, Carl-von-Ossietzky-Universität Oldenburg, 1995.

[32] M. Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems*. PhD thesis, Technische Universität Berlin, 2000.

[33] B. König and V. Kozioura. AUGUR—a tool for the analysis of graph transformation systems. *EATCS Bulletin*, 87:125–137, November 2005. Appeared in The Formal Specification Column.

[34] B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In H. Hermanns and J. Palsberg, editors, *Proceedings of TACAS '06*, volume 3920 of *Lecture Notes in Computer Science*, pages 197–211. Springer Verlag, 2006.

[35] B. König and V. Kozioura. Incremental construction of coverability graphs. *Information Processing Letters*, 103(5):203–209, 2007.

[36] B. König and V. Kozioura. AUGUR 2—a new version of a tool for the analysis of graph transformation systems. In *Proceedings of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques)*, Electronic Notes in Theoretical Computer Science. Elsevier Science, 2008.

[37] Vitali Kozioura. Verification of random graph transformation systems. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proc. of GT-VC '06 (Graph Transformation for Verification and Concurrency)*, volume 175.4 of *ENTCS*, 2006.

[38] A. Loginov, T.W. Reps, and M. Sagiv. Abstraction refinement via inductive learning. In *Proc. of CAV '05*, volume 3576 of *Lecture Notes in Computer Science*, pages 519–533. Springer Verlag, 2005.

[39] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.

[40] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, 1971.

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

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

[43] R. Milner. Bigraphical reactive systems. In K. G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR'01*, volume 2154 of *Lecture Notes in Computer Science*, pages 16–35. Springer Verlag, 2001.

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

[45] M. Pistore. *History Dependent Automata*. PhD thesis, Department of Computer Science, University of Pisa, 1999.

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

[47] A. Rensink. Model checking graph grammars. In *Proceedings of AVOCS '03 (Workshop on Automated Verification of Critical Systems)*, 2003.

[48] A. Rensink. Canonical graph shapes. In D.A. Schmidt, editor, *Proceedings of ESOP '04*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer Verlag, 2004.

[49] A. Rensink. State space abstraction using shape graphs. In *Proceedings of AVIS '04 (Third International Workshop on Automatic Verification of Infinite-State Systems)*, Electronic Notes in Theoretical Computer Science. Elsevier Science, 2004.

[50] A. Rensink and D. Distefano. Abstract graph transformation. In *Proceedings of SVV '05 (3rd International Workshop on Software Verification and Validation)*, volume 157(1) of *Electronic Notes in Theoretical Computer Science*, pages 39–59. Elsevier Science, 2005.

[51] L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.

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

[53] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(3):217–298, 2002.

[54] V. Sassone. *On the Semantics of Petri Nets: Processes, Unfolding and Infinite Computations*. PhD thesis, University of Pisa - Department of Computer Science, 1994.

[55] K. Schmidt. Distributed verification with LoLA. *Fundamenta Informaticae*, 54(2–3):253–262, 2003.

[56] G. Taentzer. AGG: A tool environment for algebraic graph transformation. In M. Nagl, A. Schürr, and M. Münch, editors, *Proceedings of AGTIVE '99 (Applications of Graph Transformations with Industrial Relevance, International Workshop)*, volume 1779 of *Lecture Notes in Computer Science*, pages 481–488. Springer Verlag, 1999.

[57] D. Varró. Towards symbolic analysis of visual modeling languages. In *Workshop on Graph Transformation and Visual Modeling Techniques '02*, volume 72 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[58] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP'97*, volume 1256 of *Lecture Notes in Computer Science*, pages 538–548. Springer Verlag, 1997.

[59] E. Yahav, T.W. Reps, S. Sagiv, and R. Wilhelm. Verifying temporal heap properties specified via evolution logic. *Logic Journal of the IGPL*, 14(5):755–783, 2006.