Unfolding Graph Transformation Systems: Theory and Applications to Verification

Dedicated to Ugo Montanari on the occasion of his 65th brithday

Paolo Baldan[1], Andrea Corradini[2], and Barbara König[3]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
[3] Abt. für Informatik und Ang. Kognitionswissenschaft, Universität Duisburg-Essen, Germany

baldan@math.unipd.it andrea@di.unipi.it barbara_koenig@uni-due.de

Abstract. The unfolding of a system represents in a single branching structure all its possible computations: it is the cornerstone both of semantical constructions and of efficient partial order verification techniques. In this paper we survey the contributions we elaborated in the last decade with Ugo Montanari and other colleagues, concerning the unfolding of graph transformation systems, and its use in the definition of a Winskel style functorial semantics and in the development of methodologies for the verification of finite and infinite state systems.

1 Introduction

Graph transformation systems (GTSs) [31] are recognized as an expressive specification formalism, especially suited for concurrent and distributed systems [16]: the (topo)logical distribution of a system can be represented naturally by using a graphical structure and the dynamics of the system, including the reconfigurations of its topology, can be modelled by means of graph rewriting rules. Moreover GTSs can be seen as a proper generalisation of a classical model of concurrency, i.e., Petri nets [29], since the latter are essentially rewriting systems on (multi)sets, the rewriting rules being the transitions.

In a research activity started under the guidance of Ugo Montanari the concurrent behaviour of GTSs has been thoroughly studied and a consolidated theory of concurrency for such systems is now available, including the generalisation of several semantics of Petri nets, like process and unfolding semantics (see, e.g., [13, 30, 7]). The unfolding construction, presented in [30] for the *single-pushout approach* and in [7] for the *double-pushout approach*, has been the basis of a functorial semantics, recently presented in [9], that generalizes to GTSs the one developed by Winskel for safe Petri nets [33]. Furthermore, building on these semantical foundations and in particular on the unfolding construction, a framework has been developed where behavioural properties of GTSs can be expressed

and verified. As witnessed, e.g., by the approaches in [25, 17] for Petri nets, truly concurrent semantics are potentially useful in the verification of finite state systems, in that they help to avoid the combinatorial explosion arising when one explores all possible interleavings of events. Such techniques have been generalized to a framework for the verification of finite state GTSs in [5]. Interestingly, several formalisms for concurrency and mobility can be encoded as GTSs, in a way that verification techniques developed for GTSs potentially carry over to such formalisms. Concurrent and mobile systems are often infinite state: in these cases we can resort to approximation techniques in order to analyze them, as proposed in [4, 10, 11].

In this paper we summarize a number of contributions published by the authors in collaboration with Ugo Montanari and other colleagues, describing a framework for the verification of systems modeled as GTSs, based on solid semantical foundations. We start by presenting the unfolding construction for GTSs in Section 2. Next we describe, in a succinct way due to size limitation, three frameworks where the unfolding construction plays a crucial role, namely the functorial semantics of [9] in Section 3, the finite prefix approach of [5] in Section 4, and the verification framework for infinite state GTSs based on finite over-approximations of the unfolding proposed in [4, 10, 11] in Section 5. Finally in Section 6 we draw some conclusions.

2 Unfolding semantics of graph transformation systems

In this section we first introduce the notion of graph rewriting used in the paper: rewriting takes place on so-called *typed graphs*, namely graphs labelled over a structure that is itself a graph [13], and it is defined according to the classical *algebraic, single-pushout approach* (see, e.g., [15]). Next we review the notion of *nondeterministic occurrence grammar*: this will be instrumental in presenting the *unfolding* of a GTS [7, 30] in Section 2.3.

2.1 Graph Transformation Systems

In the sequel, given a set A we denote by A^* the set of finite strings of elements of A. Given $u \in A^*$ we write $|u|$ to indicate the length of u. If $u = a_1 \dots a_n$ and $1 \leq i \leq n$, by $[u]_i$ we denote the i-th element a_i of u. Furthermore, if $f : A \to B$ is a function then we denote by $f^* : A^* \to B^*$ its extension to strings. When f is partial, the extension is strict, i.e., $f^*(u)$ is undefined if f is undefined on $[u]_i$ for some $i \in \{1, \dots, |u|\}$.

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

Definition 1 (graphs and graph morphisms). *A* (hyper)graph G *is a tuple* (V_G, E_G, c_G)*, where* V_G *is a set of nodes,* E_G *is a set of edges and* $c_G : E_G \to V_G^*$ *is a connection function. A node* $v \in V_G$ *is called* isolated *if it is not connected*

to any edge. Given two graphs G, G', a partial graph morphism $f : G \rightharpoonup G'$ *is a pair of partial functions $f = \langle f_V : V_G \rightharpoonup V_H, f_E : E_G \rightharpoonup E_H \rangle$ such that:*

$$c_H \circ f_E \leq f_V^* \circ c_G. \tag{1}$$

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

Notice that, according to condition (1), if f is defined over an edge then it must be defined on all its connected nodes: this ensures that the domain of f is a well-formed graph. We will write $G_1 \simeq G_2$ if G_1 and G_2 are *isomorphic*.

Definition 2 (typed graphs). *Given a graph T, a* typed graph G *over T is a pair $\langle |G|, t_G \rangle$, where $|G|$ is a graph and $t_G : |G| \to T$ is a total morphism. A* partial morphism *between T-typed graphs $f : G_1 \rightharpoonup G_2$ is a partial graph morphisms $f : |G_1| \rightharpoonup |G_2|$ consistent with the typing, i.e., such that $t_{G_1} \geq t_{G_2} \circ f$. The category of T-typed graphs and partial typed graph morphisms is denoted by T-***PGraph***.*

A typed graph G is called injective *if the typing morphism t_G is injective. More generally, given $n \in \mathbb{N}$, the graph is called* n-injective *if for any item x in T it holds that $|t_G^{-1}(x)| \leq n$, namely if the number of "instances of resources" of any type x is bounded by n.*

Given a partial typed graph morphism $f : G_1 \rightharpoonup G_2$, we denote by $dom(f)$ the domain of f typed in the obvious way.

Definition 3 (graph production and direct derivation). *Given a graph T of types, a* (T-typed graph) production q *is an injective partial typed graph morphism $L_q \overset{r_q}{\rightharpoonup} R_q$. It is called* consuming *if the morphism is not total. A production is* node preserving *if (i) r_q is total on nodes, (ii) L_q does not contain isolated nodes, and (iii) each isolated node in R_q belongs to $r_q(L_q)$. The typed graphs L_q and R_q are called the* left-hand side *and the* right-hand side *of the production, respectively.*

A match *of a production in a graph G is a total morphism $g : L_q \to G$. A match is* valid *when for any $x, y \in |L_q|$, if $g(x) = g(y)$ then $x, y \in dom(r_q)$.*

Given a production $L_q \overset{r_q}{\rightharpoonup} R_q$, a typed graph G and a match $g : L_q \to G$, we say that there is a direct derivation *$G \Rightarrow_q H$, if the diagram to the right is a pushout square in category T-***PGraph**.

$$\begin{array}{ccc} L_q & \xrightarrow{r_q} & R_q \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ G & \xrightarrow[d]{} & H \end{array}$$

Roughly speaking, the effect of the pushout construction in T-**PGraph** is the following: graph H is obtained by first deleting from the graph G the image of the items of the left-hand side which are not in the domain of r_q, namely $g(L_q - dom(r_q))$, as well as all edges that would remain dangling, and then adding the items of the right-hand side which are not in the image of r_q, namely $R_q - r_q(dom(r_q))$. The items in the image of $dom(r_q)$ are "preserved" by the rewriting step (intuitively, they are accessed in a "read-only" manner).

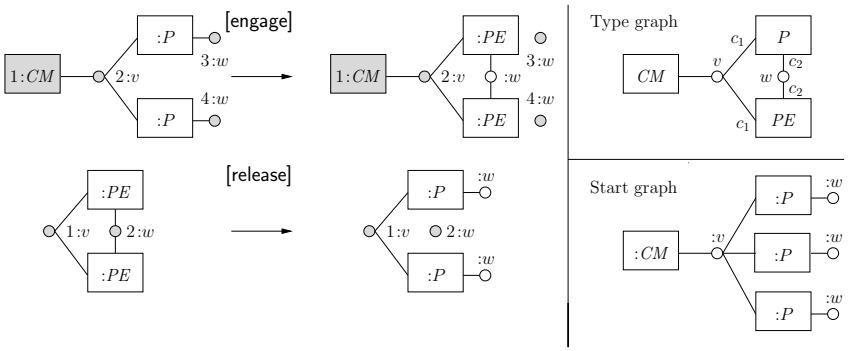

Fig. 1. The finite state GTS $\mathcal{CP}$.

Definition 4 (typed GTS and derivation). *A* (T-typed) SPO GTS $\mathcal{G}$ *(sometimes also referred to as a* (graph) grammar*) is a tuple* $\langle T, G_s, P, \pi \rangle$*, where* G_s *is the* (typed) start graph*,* P *is a set of* production names*, and* π *is a function which associates a* T*-typed production to each name in* P*. We denote by* $Elem(\mathcal{G})$ *the set* $V_T \cup E_T \cup P$*.*

A derivation *in* $\mathcal{G}$ *is a sequence of direct derivations beginning from the start graph* $\rho = \{G_{i-1} \Rightarrow_{q_{i-1}} G_i\}_{i \in \{1,\ldots,n\}}$*, with* $G_0 = G_s$*: in this case we write* $G_s \Rightarrow^*_{\mathcal{G}} G_n$*. A* T*-typed graph* G *is* reachable *in* $\mathcal{G}$ *if* $G_s \Rightarrow^*_{\mathcal{G}} G$*.*

We will consider only GTSs where all productions are *consuming*, and derivations where matches are *valid*. The restriction to consuming productions is standard in the framework of semantics combining concurrency and nondeterminism (see, e.g., [19, 33]). On the other hand, considering valid matches only is needed to have a computational interpretation which is resource-conscious, i.e., where a resource can be consumed only once. In Sections 4 and 5 we shall further restrict to node-preserving productions, for the reasons explained there.

Example 5. Consider the GTS $\mathcal{CP}$ (a variation of the running example of [5]), modeling a system where three *processes* of type P are connected to a *communication manager* of type CM (see the start graph in Fig. 1, where edges are represented as rectangles and nodes as small circles). Two processes may establish a new connection with each other via the communication manager, becoming *processes engaged* in communication (typed PE). This transformation is modelled by the production [engage] in Fig. 1: observe that a new node connecting the two processes is created. The second production [release] terminates the communication between two partners. A typed graph G over $T_{\mathcal{CP}}$ is drawn by labeling each edge or node x of G with ": $t_G(x)$". Only when the same graphical item x belongs to both the left- and the right-hand side of a production we include its identity in the label (which becomes "$x : t_G(x)$"): in this case we also shade the item, to stress that it is preserved by the production.

2.2 Nondeterministic occurrence grammars

Conceptually, a *nondeterministic occurrence grammar* $\mathcal{O}$ is a structure that can be used to provide a static description of the computations of a given GTS $\mathcal{G}$: each production of $\mathcal{O}$ represents an *event*, i.e., a specific application of a production of $\mathcal{G}$, while the items of the type graph of $\mathcal{O}$ represent items of graphs reachable in derivations of $\mathcal{G}$. Analogously to what happens for Petri nets, occurrence grammars are "safe" GTSs, where the dependency relations between productions satisfy suitable acyclicity and well-foundedness requirements. The notion of safe GTS [13] generalizes the one for P/T nets which requires that each place contains at most one token in any reachable marking.

Definition 6 (safe GTS). *A GTS $\mathcal{G} = \langle T, G_s, P, \pi \rangle$ is* safe *if, for all H such that $G_s \Rightarrow^* H$, H is injective.*

In a safe GTS, each graph G reachable from the start graph is injectively typed, and thus we can identify it with the corresponding subgraph $t_G(|G|)$ of the type graph. With this identification, a production can be applied in G only to the subgraph of the type graph which is the image via the typing morphism of its left-hand side. Thus, according to its typing, we can safely think that a production *produces*, *preserves* or *consumes* items of the type graph. Using a net-like language, we speak of *pre-set* ${}^\bullet q$, *context* $\underline{q}$ and *post-set* $q^\bullet$ of a production q, defined in the obvious way. Clearly, the items of the type graph which are used by more productions may induce certain dependencies among them: this is formalized by the *causality* and *asymmetric conflict* relations introduced next, which are pivotal for the definition of occurrence grammars.

Definition 7 (causal relation). *The* causal relation *of a grammar $\mathcal{G}$ is the binary relation $<$ over $Elem(\mathcal{G})$ defined as the least transitive relation satisfying: for any node or edge x in the type graph T, and for productions $q, q' \in P$*

1. *if $x \in {}^\bullet q$ then $x < q$;*
2. *if $x \in q^\bullet$ then $q < x$;*
3. *if $q^\bullet \cap \underline{q'} \neq \emptyset$ then $q < q'$.*

As usual $\leq$ is the reflexive closure of $<$. Moreover, for $x \in Elem(\mathcal{G})$ we denote by $\lfloor x \rfloor$ the set of causes of x in P, namely $\{q \in P : q \leq x\}$.

Note that the fact that an item is preserved by q and consumed by q', i.e., $\underline{q} \cap {}^\bullet q' \neq \emptyset$, does not imply $q < q'$. Instead, such productions are in *asymmetric conflict* (see [8, 28, 23]): The application of q' prevents q from being applied, so that when both q and q' occur in a derivation, then q must precede q'.

Definition 8 (asymmetric conflict). *The* asymmetric conflict relation *of a grammar $\mathcal{G}$ is the binary relation $\nearrow$ over the set of productions, defined by:*

1. *if $\underline{q} \cap {}^\bullet q' \neq \emptyset$ then $q \nearrow q'$;*
2. *if ${}^\bullet q \cap {}^\bullet q' \neq \emptyset$ and $q \neq q'$ then $q \nearrow q'$;*
3. *if $q < q'$ then $q \nearrow q'$.*

Condition 1 is justified by the discussion above. Condition 2 essentially expresses the fact that the ordinary symmetric conflict is encoded, in this setting, as an asymmetric conflict in both directions. Finally, since $<$ represents a global order of execution, while $\nearrow$ determines an order of execution only locally to each computation, it is natural to impose $\nearrow$ to be an extension of $<$ (condition 3).

Definition 9 ((nondeterministic) occurrence grammar). *A* (nondeterministic) occurrence grammar *is a grammar* $\mathcal{O} = \langle T, G_s, P, \pi \rangle$ *such that*

1. *its causal relation* $\leq$ *is a partial order, and, for any* $q \in P$*, the set* $\lfloor q \rfloor$ *is finite and the asymmetric conflict* $\nearrow$ *is acyclic on* $\lfloor q \rfloor$*;*
2. *the start graph* G_s *is the set* $Min(\mathcal{O})$ *of minimal elements of* $\langle Elem(\mathcal{O}), \leq \rangle$ *(with the graphical structure inherited from* T *and typed by the inclusion);*
3. *any item* x *in* T *is created by at most one production in* P*, namely* $|{}^\bullet x| \leq 1$*;*
4. *for each* $q \in P$*, the typing* t_{L_q} *is injective on the "consumed part"* $|L_q| - |dom(r_q)|$*, and* t_{R_q} *is injective on the "produced part"* $|R_q| - r_q(|dom(r_q)|)$*.*

Since the start graph of an occurrence grammar $\mathcal{O}$ is determined by $Min(\mathcal{O})$, we often do not mention it explicitly. It is possible to show that, by the defining conditions, each occurrence grammar is safe. Intuitively, conditions 1–3 recast in the framework of graph grammars the analogous conditions of occurrence nets (actually of occurrence contextual nets [8]). In particular, in condition 1, the acyclicity of asymmetric conflict on $\lfloor q \rfloor$ corresponds to the requirement of irreflexivity for the conflict relation in occurrence nets. Condition 4, instead, is closely related to safety and requires that each production consumes and produces items with multiplicity one.

The finite computations of an occurrence grammar are characterized by specific subsets of productions.

Definition 10 (configuration). *Let* $\mathcal{O} = \langle T, P, \pi \rangle$ *be an occurrence grammar. A configuration of* $\mathcal{O}$ *is a finite subset of productions* $C \subseteq P$ *such that* $\nearrow$ *is acyclic on* C*, and for any* $q \in C$*,* $\lfloor q \rfloor \subseteq C$*. Given two configurations* C *and* C' *we write* $C \sqsubseteq C'$ *if* $C \subseteq C'$ *and for any* $q \in C$*,* $q' \in C'$*, if* $q' \nearrow q$ *then* $q' \in C$*. The set of all configurations of* $\mathcal{O}$*, ordered by* $\sqsubseteq$*, is denoted by* $Conf(\mathcal{O})$*.*

The intuition that a configuration represents a computation from the start state is formalised by the next result (see Proposition 6.11 of [1]), which also provides a "static" characterisation of the graph reached by such a derivation.

Proposition 11 (reachability of graphs generated by configurations). *Let* $\mathcal{O}$ *be an occurrence grammar, let* $C \in Conf(\mathcal{O})$ *be a configuration and let*

$$gr(C) = (Min(\mathcal{O}) \cup \bigcup_{q \in C} q^\bullet) - \bigcup_{q \in C} {}^\bullet q.$$

Then $gr(C)$ *is reachable in* $\mathcal{O}$ *by applying all the productions of* C *in any order compatible with* $\nearrow$*.*

As in the case of Petri nets, reachable states can be characterized in terms of a concurrency relation: this is an easy consequence of Proposition 11.

Definition 12 (concurrent graph). *Let $\mathcal{O} = \langle T, P, \pi \rangle$ be an occurrence grammar. A subgraph G of T is called* concurrent *if (1) the asymmetric conflict $\nearrow$ restricted to $\bigcup_{x \in G} \lfloor x \rfloor$ is acyclic and finitary; and (2) $\neg(x < y)$ for all $x, y \in G$.*

Proposition 13 (concurrency vs. reachability). *Let $\mathcal{O} = \langle T, P, \pi \rangle$ be an occurrence grammar and G be a subgraph of T. Then G is concurrent iff it is a subgraph of a graph reachable in $\mathcal{O}$ by applying all the productions in $\bigcup_{x \in G} \lfloor x \rfloor$ in any order compatible with $\nearrow$.*

2.3 Unfolding construction

This section presents the unfolding construction which, applied to an SPO GTS $\mathcal{G}$, produces a nondeterministic occurrence grammar $\mathcal{U}_s(\mathcal{G})$ describing its behaviour. The idea is to begin with the start graph of the GTS, and to apply in all possible ways its productions to concurrent subgraphs, recording in the unfolding each occurrence of a production and each new graph item generated.

A basic ingredient of the construction is the *gluing* operation. It can be seen as a "partial application" of a production to a given match, in the sense that it generates the new items as specified by the production, but items that should have been deleted are not affected: intuitively, this is because such items may still be used by another production in the nondeterministic unfolding.

Definition 14 (gluing). *Let $q = r_q : L_q \rightharpoonup R_q$ be a production, G a graph and $m : L_q \to G$ a graph morphism. For any symbol z, we denote by $glue_z(q, m, G)$ the graph $\langle V, E, s, t \rangle$, where $V = V_G \cup m_z(V_{R_q})$, $E = E_G \cup m_z(E_{R_q})$, and m_z is defined by: $m_z(x) = m(x)$ if $x \in dom(r_q)$ and $m_z(x) = \langle x, z \rangle$ otherwise. The connection function and the typing are inherited from G and R_q.*

Therefore the gluing operation keeps unchanged the identity of the items already in G, and records in each newly added item from R_q the given symbol z.

The unfolding of a GTS is obtained as the union of a chain of occurrence grammars, each approximating the unfolding up to a certain causal depth.

Definition 15 (unfolding). *Let $\mathcal{G} = \langle T, G_s, P, \pi \rangle$ be a* GTS. *We inductively define, for each n, an occurrence grammar $\mathcal{U}_s(\mathcal{G})^{[n]} = \langle T^{[n]}, P^{[n]}, \pi^{[n]} \rangle$ and a pair of mappings $\varphi^{[n]} = \langle {\varphi_T}^{[n]} : T^{[n]} \to T, {\varphi_P}^{[n]} : P^{[n]} \to P \rangle$. Then the* unfolding *$\mathcal{U}_s(\mathcal{G})$ and the* folding morphism *$\varphi_{\mathcal{G}} : \mathcal{U}_s(\mathcal{G}) \to \mathcal{G}$ are the occurrence grammar and the morphism defined as the componentwise unions of $\mathcal{U}_s(\mathcal{G})^{[n]}$ and $\varphi^{[n]}$.*

($\mathbf{n = 0}$) The components of the grammar $\mathcal{U}_s(\mathcal{G})^{[0]}$ are $T^{[0]} = |G_s|$, $P^{[0]} = \pi^{[0]} = \emptyset$. Morphism $\varphi^{[0]} : \mathcal{U}_s(\mathcal{G})^{[0]} \to \mathcal{G}$ is defined by ${\varphi_T}^{[0]} = t_{G_s}$, ${\varphi_P}^{[0]} = \emptyset$.

($\mathbf{n \to n + 1}$) The occurrence grammar $\mathcal{U}_s(\mathcal{G})^{[n+1]}$ is obtained by extending $\mathcal{U}_s(\mathcal{G})^{[n]}$ with all the possible production applications to concurrent subgraphs of its type graph. More precisely, let $M^{[n]}$ be the set of pairs $\langle q, m \rangle$ such that $q \in P$ is a production in $\mathcal{G}$, $m : L_q \to \langle T^{[n]}, {\varphi_T}^{[n]} \rangle$ is an injective match and $m(|L_q|)$ is a concurrent *subgraph of $T^{[n]}$. Then $\mathcal{U}_s(\mathcal{G})^{[n+1]}$ is the occurrence grammar resulting after performing the following steps for each $\langle q, m \rangle \in M^{[n]}$.*

– *Add to* $P^{[n]}$ *the pair* $\langle q, m\rangle$ *as a new production name and extend* $\varphi_P{}^{[n]}$ *so that* $\varphi_P{}^{[n]}(\langle q, m\rangle) = q$. *Intuitively,* $\langle q, m\rangle$ *represents an occurrence of* q, *where the match* m *is needed to record the "history".*
– *Extend the type graph* $T^{[n]}$ *by adding to it a copy of each item generated by the application* q, *marked by* $\langle q, m\rangle$ *(in order to keep trace of the history). The morphism* $\varphi_T{}^{[n]}$ *is extended consequently. Formally, the* T*-typed graph* $\langle T^{[n]}, \varphi_T{}^{[n]}\rangle$ *is replaced by* $glue_{\langle q,m\rangle}(q, m, \langle T^{[n]}, \varphi_T{}^{[n]}\rangle)$.
– *The production* $\pi^{[n]}(\langle q, m\rangle)$ *has the same untyped components as* $\pi(q)$. *The typing of the left-hand side is determined by* m, *and each item* x *in* $|R_q| - r_q(|dom(r_q)|)$ *is typed over the new item* $\langle x, \langle q, m\rangle\rangle$ *of the type graph.*

The most relevant property of the unfolding is the fact that it provides a compact representation of the behaviour of $\mathcal{G}$, and in particular it represents all the graphs reachable in $\mathcal{G}$, in the following sense. If T' is the type graph of the unfolding of $\mathcal{G}$, $\varphi_T : T' \to T$ is the type graph component of the folding morphism, and G is a subgraph of T', let us denote by $\varphi_T(G)$ the same graph, but typed over T by the restriction of the folding morphism, i.e., $\varphi_T(G) = \langle G, \varphi_{T|G}\rangle$. Then the next result is an easy consequence of the characterization of the unfolding as a right adjoint, shown in [9].

Theorem 16 (completeness of the unfolding). *Let* $\mathcal{G} = \langle T, G_s, P, \pi\rangle$ *be a* GTS. *A* T*-typed graph* G *is reachable in* $\mathcal{G}$ *iff there exists a configuration* C *in* $Conf(\mathcal{U}(\mathcal{G}))$ *such that* $G \simeq \varphi_T(gr(C))$.

3 Functorial semantics: from nets to SPO grammars

In this section we discuss the role played by the unfolding construction in the development of a functorial semantics, first for Petri nets and then for GTSs.

3.1 A coreflective semantics for Petri nets

In the theory of Petri Nets, the unfolding construction, whose generalization to SPO grammars has been presented in the previous section, is the cornerstone of a functorial semantics which has been developed by Winskel in [33], based on previous works with Nielsen and Plotkin [27]. Winskel shows that there is a chain of categorical coreflections (a special kind of adjunction), leading from the category **S-N**, having safe (marked) P/T nets as objects and suitably defined morphisms, to the category **Dom** of finitary prime algebraic domains, through the categories **O-N** of occurrence nets and **PES** of prime event structures (PESs).

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

The first step is the construction unwinding a safe net $N \in \mathbf{S\text{-}N}$ into its unfolding $\mathcal{U}(N)$ which, as in the case of grammars, records in its branching

and acyclic structure all the possible computations of the original net N. Every possible transition occurrence (*event*) is identified uniquely in the unfolding by its *history*, i.e., by the finite set of events which caused it, and events are related by the causality and *symmetric* conflict relations induced by the intersections of the pre- and post-sets: differently from the case of GTSs, in a (safe) Petri net all conflicts are symmetric because transitions do not have a context. Functor $\mathcal{U}(N) : \mathbf{S\text{-}N} \to \mathbf{O\text{-}N}$ is the right adjoint to the inclusion functor $\mathbf{O\text{-}N} \hookrightarrow \mathbf{S\text{-}N}$.

The subsequent step abstracts an occurrence net O to a PES $\mathcal{E}(O)$, which is obtained from the unfolding simply by forgetting the places, and remembering only the events and the causality and conflict relations among them. From a prime event structure E it is possible to generate freely an occurrence net $\mathcal{N}(E)$ which is the "most general" among those having E as underlying PES. Such a net is obtained by considering the events of E as transition occurrences, and introducing, among others, one fresh place for every pair of events related by causality or conflict in E, in order to enforce the same relationships in $\mathcal{N}(E)$. This construction defines a left adjoint to functor $\mathcal{E} : \mathbf{PES} \to \mathbf{O\text{-}N}$. The last step, which establishes an equivalence between the categories **PES** and **Dom**, maps any event structure to its domain of configurations.

3.2 Coreflective semantics: From nets to SPO grammars

During several years the first two authors cooperated with Ugo Montanari in a project aimed at generalizing the coreflective semantics of nets to graph grammars. At the beginning, most of the efforts were concentrated on the *double-pushout approach* to graph transformation, and partial results were reported in [1, 7]. Only quite recently, however, a complete Winskel's style coreflective semantics has been developed successfully for the SPO approach, as reported in [9], and summarized by the following chain of adjunctions:

$$\mathbf{SPO\text{-}GG} \underset{\mathcal{U}_s}{\overset{\perp}{\hookleftarrow}} \mathbf{SPO\text{-}OG} \underset{\mathcal{E}_s}{\overset{\mathcal{N}}{\underset{\perp}{\leftarrow}}} \mathbf{AES} \underset{\mathcal{L}_a}{\overset{\mathcal{P}_a}{\underset{\perp}{\leftarrow}}} \mathbf{Dom}$$

Without delving into technical details, we summarize here the most challenging problems we had to address during our project, and the way we faced them.

Obviously, the starting point is Definition 15, describing the unfolding construction at the level of objects. To extend it to a functor $\mathcal{U}_s$ providing a right adjoint to the inclusion of the category **SPO-OG** of occurrence grammars into the category **SPO-GG** of general grammars, we first had to identify a sensible definition of grammar morphism. The chosen notion, discussed in [1], is more general than others proposed in the literature, and, unlike others, it coincides with the Petri net morphisms of [33] when restricted to graph grammars which represent Petri nets. Furthermore, inspired by corresponding results for nets in [26], we considered *semi-weighted* grammars, a class that strictly includes safe grammars, with the additional advantage of being characterized by a "static condition", not involving the behaviour but just the structure of the grammar.

Concerning the next adjunction in the chain, it is worth stressing here the presence of the category of *asymmetric* event structures (**AES**) in place of **PES**.

The point is that *prime* event structures, which only include the causality and *symmetric* conflict relations, are not sufficient to capture properly the dependencies among events of systems where productions may have a context, modeling the read-only access to resources. In these cases, which include SPO grammars and *contextual Petri nets*, asymmetric conflicts (see Definition 8) arise as a primitive concept. This motivated the introduction of asymmetric event structures (which are equipped with causality and *asymmetric* conflict), and the study of their category **AES**, which is shown to contain **PES** as a coreflective subcategory (see [8]). It is worth mentioning that, with the goal of providing an event structure semantics for nominal calculi, in [12] a simpler functorial semantics is presented for a restricted class of *persistent* grammars, for which category **PES** turns out to be sufficient, because asymmetric conflicts cannot arise.

Given an occurrence grammar $\mathcal{O}$, the AES $\mathcal{E}_s(\mathcal{O})$ is obtained by considering the productions as events, equipped with causality and asymmetric conflict as for Definitions 7 and 8. Moreover, a construction inspired by the work on contextual nets [8] allows one to build a canonical occurrence SPO graph grammar $\mathcal{N}(A)$ from a given asymmetric event structure A, providing a left-adjoint to functor $\mathcal{E}_s$ (technically, this works when dealing with injective matches only).

The chain of coreflection is completed by using the fact that the equivalence between **PES** and **Dom** generalizes to a coreflection between **AES** and **Dom** [8].

4 Verification of finite state GTSs

In the approach originally proposed by McMillan for the analysis of Petri nets [25] and further developed by many authors (see, e.g., [17, 18, 32]) the idea is that given a finite state net, it is possible to identify a *finite* fragment of its unfolding which is *complete*, i.e., which provides full information about the system as far as reachability properties are concerned: this fragment can be characterized as the maximal prefix of the unfolding not including *cut-off events*, i.e., transitions which do not contribute to generating new markings.

In this section we summarize [5], where by exploiting the unfolding construction of Section 2.3, we have generalized McMillan's approach to SPO GTSs by introducing an original notion of "strong cut-off" (which takes into account the fact that a production can have several different histories), and we have shown how a finite complete prefix of the unfolding can be used to verify interesting properties of the graphs reachable in the GTS.

4.1 Rewriting up to isolated nodes

In the work on verification of graph transformation systems summarized in this and in the next section, we consider only systems consisting of node-preserving productions, as introduced in Definition 3, and rewriting *up to isolated nodes* (graphs which are isomorphic after deleting all isolated nodes are considered indistinguishable). As far as the expressive power is concerned, this is a mild restriction, since the deletion of a node can usually be modelled by leaving it

isolated: Conditions (ii) and (iii) of Definition 3 guarantee that isolated nodes do not take part to the rewriting. Also, this is consistent with the fact that the logic we shall use for verification purposes (see Definition 22) is not able to distinguish graphs which are isomorphic up to isolated nodes.

In the rest of the paper, we will assume that all productions are node preserving. Moreover, given any graph G and any subset of edges $X \subseteq E_G$, we denote by $graph(X)$ the smallest subgraph of G having X as set of edges, and we say that G and G' are *isomorphic up to isolated nodes*, denoted $G \,\ddot{\simeq}\, G'$, if $graph(E_G) \simeq graph(E_{G'})$. Finally, for a fixed $n \in \mathbb{N}$, we say that a GTS $\mathcal{G}$ is *n-bounded* if for each graph H reachable in $\mathcal{G}$ there is an n-injective graph H' such that $H' \,\ddot{\simeq}\, H$, and a GTS is *bounded* or *finite state* if it is n-bounded for some $n \in \mathbb{N}$.

4.2 Finite complete prefix of bounded GTSs

A *history* of a production in a computation is the set of all the events which must precede its application. Due to the presence of asymmetric conflicts, a production q does not have a unique history in general, because depending on the specific computation we consider, some of the productions in asymmetric conflict with q might have been applied or not before q.

Definition 17 (history). *Let $\mathcal{O}$ be an occurrence grammar, let $C \in Conf(\mathcal{O})$ be a configuration and let $q \in C$. The* history *of q in C is the set of events $C[\![q]\!] = \{q' \in C : q' \nearrow_C^* q\}$, where $\nearrow_C$ is the restriction of $\nearrow$ to C. We denote by $Hist(q)$ the set of histories of q, i.e., $Hist(q) = \{C[\![q]\!] : C \in Conf(\mathcal{O})\}$.*

Now, let $\mathcal{G} = \langle T, G_s, P, \pi \rangle$ denote a GTS, fixed throughout the section, and let $\mathcal{U}_s(\mathcal{G}) = \langle T', P', \pi' \rangle$ be its unfolding with $\varphi : \mathcal{U}_s(\mathcal{G}) \to \mathcal{G}$ the folding morphism, as in Definition 15. In order to identify a finite and complete prefix of the unfolding of a bounded GTS, the idea is to avoid useless productions in the unfolding, i.e., productions which do not contribute to generating new graphs. The definition of "cut-off event" introduced by McMillan for Petri nets in order to formalize such a notion has to be adapted to this context since, as explained above, for graph grammars a production may have different histories.

Definition 18 (cut-off). *A production $q \in P'$ of the unfolding $\mathcal{U}_s(\mathcal{G})$ is a* cut-off *if there exists $q' \in P'$ such that $\varphi_T(gr(\lfloor q \rfloor)) \,\ddot{\simeq}\, \varphi_T(gr(\lfloor q' \rfloor))$ and $||\lfloor q' \rfloor|| < ||\lfloor q \rfloor||$.*

A production q is a strong cut-off *if for all $C_q \in Hist(q)$ there exist $q' \in P'$ and $C_{q'} \in Hist(q')$ such that $\varphi_T(gr(C_q)) \,\ddot{\simeq}\, \varphi_T(gr(C_{q'}))$ and $|C_{q'}| < |C_q|$. The* truncation *of $\mathcal{G}$ is the greatest prefix $\mathcal{T}(\mathcal{G})$ of $\mathcal{U}_s(\mathcal{G})$ not including strong cut-offs.*

Theorem 19 (completeness and finiteness of the truncation). *The truncation $\mathcal{T}(\mathcal{G})$ is a complete prefix of the unfolding, i.e., for any reachable graph G of $\mathcal{G}$ there is a configuration C in $Conf(\mathcal{T}(\mathcal{G}))$ such that $\varphi_T(gr(C)) \,\ddot{\simeq}\, G$. Furthermore, if $\mathcal{G}$ is bounded then the truncation $\mathcal{T}(\mathcal{G})$ is finite.*

Unfortunately, neither the statement of the above theorem nor its proof (see Appendix B of the full version of [5]) suggest a way to modify the unfolding construction of Definition 15 in order to obtain the truncation of a bounded GTS: this is because the notion of strong cut-off refers to the set of histories of a production, that in general could be infinite. Only recently the authors proposed an algorithm (see [6]) which solves a similar problem in the simpler case of *contextual* Petri nets: we are confident that this algorithm can be adapted to GTSs, but space constraints do not allow us to elaborate on that here.

Instead, a class of GTSs can be identified for which an obvious adaptation of the unfolding construction does produce a finite complete prefix. It is characterized by a property called "read-persistence", since it appears as the graph grammar theoretical version of the notion introduced for contextual nets in [32].

Definition 20 (read-persistence). *An occurrence grammar $\mathcal{O} = \langle T, P, \pi \rangle$ is called* read-persistent *if for any $q_1, q_2 \in P$, if $q_1 \nearrow q_2$ then either q_1 is a cause of q_2, or q_1 and q_2 are in* conflict*, i.e., they cannot fire in the same derivation. A* GTS $\mathcal{G}$ *is called* read-persistent *if its unfolding $\mathcal{U}(\mathcal{G})$ is read-persistent.*

An adaptation of the algorithm originally proposed in [25] for ordinary nets and extended in [32] to read-persistent contextual nets, works for read-persistent GTSs as well, because in this case every production has a single history, and thus the notions of cut-off and of strong cut-off of Definition 18 coincide. Roughly, for such GTSs a complete finite prefix can be obtained by slightly modifying the inductive step of the construction of Definition 15 as follows: the production occurrences in the set $M^{[n]}$ have to be handled in increasing order according to the size of the corresponding history, and a production occurrence has to be added to the unfolding only if it is not a cut-off in the prefix computed so-far.

An obvious class of read-persistent systems consists of those GTSs where any edge preserved by productions is never consumed. For instance, the GTS $\mathcal{CP}$ in our running example is read-persistent, since CM, the only edge preserved by productions, is never consumed. Its truncation is the grammar $\mathcal{T}(\mathcal{CP})$ depicted in Fig. 2. Denote by $T_{\mathcal{T}}$ its type graph. Note that applying the production [release] to any subgraph of $T_{\mathcal{T}}$ matching its left-hand side would result in a cut-off: this is the reason why $\mathcal{T}(\mathcal{CP})$ does not include any instance of production [release]. The start graph of the truncation is isomorphic to the start graph of GTS $\mathcal{CP}$ and it is mapped injectively to the graph of types $T_{\mathcal{T}}$ in the obvious way.

4.3 Checking properties of reachable graphs

Given a finite state GTS $\mathcal{G}$, a complete prefix can be used to check whether there exists at least one reachable graph satisfying a certain property F, or if a property F is an "invariant" of $\mathcal{G}$, i.e., it is satisfied by all reachable graphs. The graph properties of interest will be expressed as formulae of a quite expressive logic called $\mathcal{L}2$ (introduced below), whose induced logical equivalence on finite graphs is "isomorphism up to isolated nodes". That is, two finite graphs G and G' satisfy exactly the same formulae of $\mathcal{L}2$ if and only if $G \ddot{\cong} G'$.

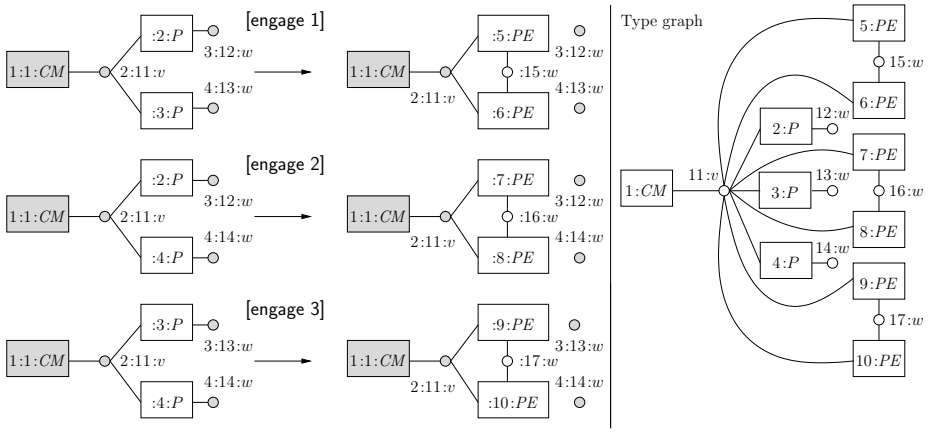

Fig. 2. The truncation $\mathcal{T}(\mathcal{CP})$ of the GTS in Fig. 1.

Now, the usefulness of the truncation (or of any other finite complete prefix) resides in the fact that since for each graph G reachable in $\mathcal{G}$ there is a G' reachable in $\mathcal{T}(\mathcal{G})$ such that $\varphi_T(G') \ddot{\simeq} G$, it is sufficient to consider the graphs reachable in $\mathcal{T}(\mathcal{G})$, retyped over T via $\varphi_T(\cdot)$. But we know that $\mathcal{T}(\mathcal{G})$ is an occurrence grammar, and thus the reachable graphs can be identified with subgraphs of its type graph, which in turn are uniquely identified by their sets of edges, because we rewrite up to isolated nodes.

Therefore, we can verify if F holds for all (some) reachable graphs by checking that $\varphi_T(\mathit{graph}(m)) \models F$ for all (some) sets of edges m reachable in $\mathcal{T}(\mathcal{G})$. This fact can be formalized in a convenient way by introducing a (safe) Petri net "underlying" the truncation, and seeing a set of edges of $\mathcal{T}(\mathcal{G})$ as a marking of such net. Furthermore, since this net is fixed and finite, it is possible to translate every formula $F \in \mathcal{L}2$ into a propositional formula over markings $M[F]$ such that, for any reachable marking m,

$$\varphi_T(\mathit{graph}(m)) \models F \quad \text{iff} \quad m \models M[F].$$

In this way the original problem is reduced to a verification problem of a formula over a Petri net, for which existing tools could be used.

Given an occurrence grammar $\mathcal{O}$, the underlying Petri net is an occurrence contextual net (i.e., a Petri net with read arcs, see, e.g., [8, 32]).

Definition 21 (Petri net underlying an occurrence grammar). *The* contextual Petri net underlying *an occurrence grammar $\mathcal{O} = \langle T', P', \pi' \rangle$, denoted by $\mathsf{Net}(\mathcal{O})$, is the safe Petri net having the set of edges $E_{T'}$ as places and a transition for every production $q \in P'$, with pre-set ${}^\bullet q \cap E_{T'}$, post-set $q^\bullet \cap E_{T'}$ and context $\underline{q} \cap E_{T'}$.*

For instance, the Petri net $\mathsf{Net}(\mathcal{T}(\mathcal{CP}))$ underlying the truncation of $\mathcal{CP}$ (see Fig. 2) is depicted in Fig. 3. Read arcs are represented as undirected lines.

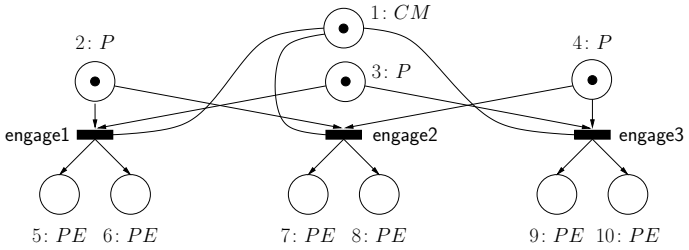

Fig. 3. The Petri net underlying the truncation $\mathcal{T}(\mathcal{CP})$ in Fig. 2

Next we define the monadic second-order logic $\mathcal{L}2$ used for specifying properties of graphs typed over a fixed graph T. Quantification is allowed over edges, but not over nodes (as in [14]).

Definition 22 (graph logic). *Let $\mathcal{X}_1 = \{x, y, \ldots\}$ be a set of (first-order) edge variables and let $\mathcal{X}_2 = \{X, Y, \ldots\}$ be a set of (second-order) variables representing edge sets. The set of* graph formulae *of the logic $\mathcal{L}2$ is defined as follows, where $\ell \in E_T$, $i, j \in \mathbb{N}$:*

$$\begin{array}{rll} F ::= & x = y \mid c_i(x) = c_j(y) \mid \mathit{type}(x) = \ell \mid x \in X & \textit{(Predicates)} \\ & F \vee F \mid \neg F \mid \exists x.F \mid \exists X.F & \textit{(Connectives / Quantifiers)} \end{array}$$

We denote by $\mathit{free}(F)$ and $\mathit{Free}(F)$ the sets of first-order and second-order variables, respectively, occurring free in F, defined in the obvious way.

Given a T-typed graph G, a formula F in $\mathcal{L}2$, and two valuations $\sigma : \mathit{free}(F) \to E_{|G|}$ and $\Sigma : \mathit{Free}(F) \to \mathcal{P}(E_{|G|})$, the *satisfaction relation* $G \models_{\sigma,\Sigma} F$ is defined inductively in the usual way; for instance $G \models_{\sigma,\Sigma} x = y$ iff $\sigma(x) = \sigma(y)$, $G \models_{\sigma,\Sigma} \mathit{type}(x) = \ell$ iff $t_G(\sigma(x)) = \ell$, and $G \models_{\sigma,\Sigma} x \in X$ iff $\sigma(x) \in \Sigma(X)$.

Interesting graph properties can be expressed in $\mathcal{L}2$, such as the existence or adjacency of edges with specific typing, and the absence of certain paths or of certain cycles. Such properties may be used to represent in the graph transformation model relevant properties of the system at hand, such as security properties or deadlock-freedom.

Recall that a *marking* of a safe net is simply a subset of its places. The syntax of the formulae over markings is

$$Q ::= e \mid \neg Q \mid Q \wedge Q \mid Q \vee Q \mid \mathsf{true} \mid \mathsf{false},$$

where $e \in E_{T'}$. These formulae are interpreted over markings of $\mathsf{Net}(\mathcal{T}(\mathcal{G}))$: $m \models e$ if $e \in m$, and logical connectives are treated as usual.

As mentioned above, given a GTS $\mathcal{G}$ and the truncation $\mathcal{T}(\mathcal{G})$ (or any other a finite complete prefix), any formula $F \in \mathcal{L}2$ can be effectively translated to a marking formula $M[F]$ such that $\mathcal{G}$ satisfies F iff the Petri net $\mathsf{Net}(\mathcal{T}(\mathcal{G}))$ underlying the prefix satisfies $M[F]$. We omit the details of the translation,

which can be found in [5], and we focus on the running example $\mathcal{CP}$. Suppose, e.g., that we want to check that all graphs reachable in our sample GTS $\mathcal{CP}$ satisfy Φ, where Φ is a $\mathcal{L}2$ formula specifying that every engaged process is connected through connection c_2 to exactly one other engaged process, i.e.,

$$\Phi \equiv \forall x.(type(x) = PE \Rightarrow \exists y.(x \neq y \wedge type(y) = PE \wedge c_2(x) = c_2(y) \\ \wedge \forall z.(type(z) = PE \wedge x \neq z \wedge c_2(x) = c_2(z) \Rightarrow y = z))).$$

The encoding procedure leads to the formula $Q = M[\Phi]$

$$Q \equiv (5\colon PE \iff 6\colon PE) \wedge (7\colon PE \iff 8\colon PE) \wedge (9\colon PE \iff 10\colon PE)$$

which has to be checked for all reachable markings of the net of Fig. 3. An efficient algorithm for checking if a marking formula is satisfied by at least one reachable marking of an (occurrence) net $\mathsf{Net}(\mathcal{T}(\mathcal{G}))$ is presented in [5]: it exploits the mutual relationships between items expressed by the causality, (asymmetric) conflict and concurrency relations.

5 Verification of infinite state GTSs

If a GTS is not finite state, obviously no finite prefix of the unfolding can be complete in the sense of Theorem 19. In this section we describe a framework, developed in [4, 10, 11], where behavioural properties of systems described as (possibly infinite state) GTSs can be specified and verified. Here we consider rewriting up to isolated nodes (see Section 4.1), and further we require matches to be injective on edges.

Following the guidelines of the verification technique presented in the previous section, the framework is based on finite approximations of the unfolding of a given GTS which have an underlying Petri net structure. On these structures, formulae of a suitable temporal logic interpreted over derivations of a GTS can be verified, by first translating them to a simpler logic describing computations of a fixed Petri net.

5.1 Approximating the behaviour of GTSs.

A basic ingredient of our verification framework is a technique, proposed in [4, 10], for approximating the behaviour of GTSs by means of finite Petri net-like structures. More precisely, an *approximated unfolding* construction allows to generate from a given GTS $\mathcal{G}$ (which can be infinite state) suitable finite structures, called *coverings*, which provide (over-)approximations of the behaviour of $\mathcal{G}$.

The coverings of a GTS $\mathcal{G}$ are *Petri graphs over* $\mathcal{G}$, i.e., (contextual) Petri nets equipped with an additional graphical structure where the places play the role of edges, while the transitions represent applications of the productions of $\mathcal{G}$.

In the following, given a set A we denote by $A^{\oplus}$ the free commutative monoid generated by A, whose elements are finite multisets of elements of A. If $f : A \to B$ is a function, then we denote by $f^{\oplus}\colon A^{\oplus} \to B^{\oplus}$ its extension to multisets.

Definition 23 (Petri graph). *Let $\mathcal{G} = \langle T, G_0, P, \pi \rangle$ be a* GTS. *A* Petri graph *K* (over $\mathcal{G}$) *is a tuple $\langle G, N, m_0 \rangle$ where G is a T-typed graph and*

- *$N = \langle E_G, T_N, {}^\bullet(\ ), (\ )^\bullet, \underline{(\ )}, p_N \rangle$ is a place/transition Petri net, where the set of places E_G is the set of edges of G, T_N is the set of transitions, ${}^\bullet(\ ), (\ )^\bullet, \underline{(\ )} \colon T_N \to E_G^\oplus$ specify the pre-set, post-set and context of each transition and $p_N \colon T_N \to P$ is the labelling function, mapping each transition to a corresponding production;*
- *$m_0 \in E_G^\oplus$ is the* initial marking *of the Petri graph, satisfying $m_0 = \iota^\oplus(E_{G_0})$ for a suitable graph morphism $\iota : G_0 \to G$ (i.e., m_0 must properly correspond to the initial state of $\mathcal{G}$).*

We will write $m\,[q\rangle\,m'$ if a transition labelled by $q \in P$ is enabled at marking m and its firing produces m'. A marking is called reachable (coverable) *in K if it is reachable (coverable) from the initial marking in the Petri net underlying K.*

As an example, let $\mathcal{U}_s(\mathcal{G}) = \langle T', P', \pi' \rangle$ be the unfolding of a GTS $\mathcal{G} = \langle T, G_s, P, \pi \rangle$, and let $\langle \varphi_T \colon T' \to T, \varphi_P \colon P' \to P \rangle$ be the folding morphism, as presented in Definition 15. Then it is possible to see the unfolding as a Petri graph $\langle G, N, m_0 \rangle$ for $\mathcal{G}$: the net component N is as for Definition 21, the labeling of transitions is given by φ_P, G is the T-typed graph $\langle T', \varphi_T \rangle$, and m_0 is the set of minimal edges, with respect to causality, of G.

The coverings of a GTS $\mathcal{G}$ can approximate its behaviour at different levels of accuracy. For each $k \in \mathbb{N}$, the *k-covering* of $\mathcal{G}$, denoted $\mathcal{C}^k(\mathcal{G})$, over-approximates the behaviour of $\mathcal{G}$ in the sense that every derivation sequence of $\mathcal{G}$ is mapped to a valid firing sequence of (the Petri net component of) $\mathcal{C}^k(\mathcal{G})$, and every graph reachable from the start graph of $\mathcal{G}$ can be mapped homomorphically to (the graphical component of) $\mathcal{C}^k(\mathcal{G})$, and its image is reachable in the Petri graph. Furthermore, this over-approximation is exact up to causal depth k, in the sense that each graph reachable in $\mathcal{G}$ in at most k derivation steps can be mapped *injectively* to $\mathcal{C}^k(\mathcal{G})$ (see Section 5.2).

The algorithm for the construction of the k-covering of a GTS $\mathcal{G}$ works inductively like the unfolding construction, but the classical *unfolding steps*, where the application of a production to a given match is recorded by adding to the type graph the newly generated items and to the set of productions the new production occurrence, are interleaved with suitably defined *folding steps*, which merge in the graphical part of the current Petri graph two occurrences of the left-hand side of a production, if one causally depends on the other. The termination of the algorithm is ensured by giving higher priority to folding steps, and the exactness of the approximation up to causal depth k by forbidding the application of folding steps to items of smaller depth.

We define the *depth of a transition t* (an element of $\mathbb{N} \cup \{\infty\}$) to be the length of the longest sequence $t_0 < t_1 < \ldots < t_n < t$. The depth of an edge is the maximum among the depths of transitions which contain the edge in their post-set. The k-covering $\mathcal{C}^k(\mathcal{G})$ of a GTS $\mathcal{G} = \langle T, G_0, P, \pi \rangle$ is produced by the last step of the following (terminating) algorithm which generates a sequence $K_i = \langle G_i, N_i, m_i \rangle$ of Petri graphs over $\mathcal{G}$.

1. $K_0 = \langle G_0, N_0, m_0 \rangle$, where the net N_0 contains no transitions and $m_0 = E_{G_0}$.
2. As long as one of the following steps is applicable, transform K_i into K_{i+1}, giving precedence to folding steps.

   Unfolding. Find a production $q \in P$ with $\pi(q) : L_q \rightharpoonup R_q$ and a match $n\colon L_q \to G_i$ such that $n^{\oplus}(E_{L_q})$ is coverable in K_i. Then extend K_i by gluing R_q to G_i (as described in Definition 14) and add a new transition, labelled by q, representing the application of production q.

   Folding. Find a production $q \in P$ with $\pi(q) : L_q \rightharpoonup R_q$ and two matches $n, n'\colon L_q \to G_i$, at depth greater than or equal to k, such that
   - $n^{\oplus}(E_L)$ and $n'^{\oplus}(E_L)$ are coverable in N_i and
   - the first match has been unfolded with the introduction of a transition t and the second match causally depends on t.

   Then merge the two matches, by setting $n(e) \equiv n'(e)$ for each $e \in E_{L_q}$, and factoring all components of K_i through the equivalence relation induced by $\equiv$ on edges, nodes and transitions.

For example, if we extend the GTS $\mathcal{CP}$ of Example 5 with production [fork] (see Fig. 4) that models the forking of a non-engaged process, we obtain an infinite state system: in fact graphs with an unbounded number of processes are reachable. If we compute the coarsest approximation, i.e., the 0-covering, we obtain the Petri graph shown in Fig. 4 (edge and node identities are omitted):

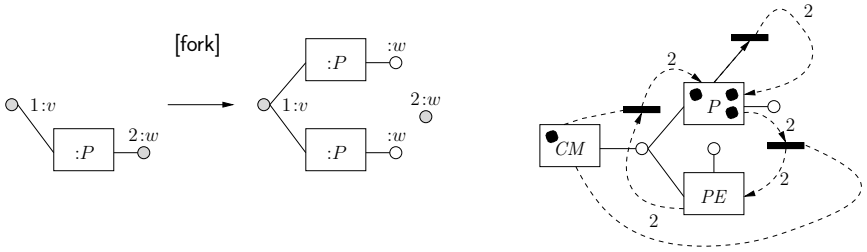

Fig. 4. Additional rule [fork] (left) and Petri graph over-approximating the GTS (right)

Several structures have been merged, for instance all engaged and all non-engaged processes. This happens because all three initial processes will be the cause of another future process, causing the fusion of all of them.

Despite the fact that the graph underlying the Petri graph is not very different from the type graph of the GTS, there are still some interesting properties of the system that can be proved by exploiting the 0-covering, using the techniques described in the next section:

- There is always exactly one communication manager.
- There will always be at least three processes (engaged or non-engaged).
- The number of engaged processes is always even.

– No engaged process is ever connected to a non-engaged process.

Due to the simplicity of the running example, these properties could easily be proved also as invariants of the transformation rules. For more complex examples we refer the reader to [4, 2].

5.2 Verifying behavioural properties of GTSs

As mentioned above, the k-covering $\mathcal{C}^k(\mathcal{G})$ over-approximates the behaviour of the original GTS $\mathcal{G}$. In order to formalize this fact, we will first generalize the notion of a subgraph generated by a set of edges (introduced at the end of Section 4.1) to a graph generated by a marking: Let $\langle G, N, m_0\rangle$ be a Petri graph and let $m \in E_G^{\oplus}$ be a marking of N. The *graph generated* by m, denoted by $graph(m)$, is the T-typed graph H without isolated nodes (which is unique up to isomorphism) such that there exists a T-typed morphism $\psi\colon H \to G$ injective on nodes with $\psi^{\oplus}(E_H) = m$.

Proposition 24 (simulation). *Let* $\mathbf{G}$ *be the set of graphs reachable from* G_0 *in* $\mathcal{G}$ *and let* $\mathbf{M}$ *be the set of reachable markings in* $\mathcal{C}^k(\mathcal{G}) = \langle G, N, m_0\rangle$*. Then there exists a* simulation $S \subseteq \mathbf{G} \times \mathbf{M}$ *with the following properties:*

- $(G_0, m_0) \in S$*;*
- *whenever* $(G', m') \in S$ *and* $G' \Rightarrow_q G''$*, then there exists a marking* m'' *with* $m' [q\rangle m''$ *and* $(G'', m'') \in S$*;*
- *for every* $(G', m') \in S$ *there exists an* edge-bijective *graph morphism* $\varphi\colon G' \to graph(m')$*.*

The simulation relation just described, whose existence can be proved fairly easily by construction, allows one to exploit the finite k-covering $\mathcal{C}^k(\mathcal{G})$ to verify certain properties of the reachable graphs of GTS $\mathcal{G}$. In fact, if a given property over graphs $F \in \mathcal{L}2$ is *reflected* by edge-bijective graph morphisms (i.e., if $f : G \to G'$ is edge-bijective and $G' \models F$ then $G \models F$), then if F is satisfied by graph $graph(m)$ for all markings m reachable in $\mathcal{C}^k(\mathcal{G})$, it is also satisfied by all graphs reachable in $\mathcal{G}$.

A couple of considerations are in order here. First, unfortunately it is undecidable if a formula of $\mathcal{L}2$ is reflected by edge-bijective morphisms, but a syntactic, sufficient criterion based on a simple type system is presented in [11]. Second, the Petri net underlying the k-covering is finite, but in general it is not finite state. Nevertheless, several verification techniques and tools have been developed for the analysis of nets, and thus the possibility of reducing the verification of a property from reachable graphs of a GTS to reachable markings of a net is of high pragmatic value.

To this aim, following the guidelines we have described in Section 4.3 for finite state GTSs, first we introduced *multiset formulae* which are evaluated on markings of the k-coverings: their syntax is obtained by extending the one presented in Section 4.3 with the atomic formula $\#e \leq c$ for $e \in E_G$ and $c \in \mathbb{N}$, meaning *the number of tokens in* e *is smaller than or equal to* c. Next we

have provided an encoding M_2 of $\mathcal{L}2$-formulae into multiset formulae, such that $graph(m) \models F \iff m \models M_2[F]$ for every reachable marking of $\mathcal{C}^k(\mathcal{G})$. This translation is a kind of quantifier elimination procedure, which is possible because the graph underlying $\mathcal{C}^k(\mathcal{G})$ is finite.

Finally, we enriched the verification framework with a temporal logic called $\mu\mathcal{L}2$, which is a propositional μ-calculus where atomic propositions are formulae of $\mathcal{L}2$. The formulae of $\mu\mathcal{L}2$ are interpreted over a *graph transition system*, i.e., a transition system where the states are graphs, and their syntax is the following:

$$M ::= F \mid X \mid \Diamond M \mid \Box M \mid \neg M \mid M_1 \vee M_2 \mid M_1 \wedge M_2 \mid \mu X.M \mid \nu X.M$$

where F ranges over closed formulae in $\mathcal{L}2$ and $X \in \mathcal{X}$ are proposition variables. Intuitively, an atomic proposition $F \in \mathcal{L}2$ holds in any state (graph) satisfying F according to the discussion after Definition 22. A formula $\Diamond M$ / $\Box M$ holds in a state if some / any single step leads to a state where M holds. The connectives $\neg$,$\vee$, $\wedge$ are interpreted in the usual way, and the formulae $\mu X.M$ and $\nu X.M$ represent the *least* and *greatest fixed point* over X, respectively.

Now, for suitable fragments of logic $\mu\mathcal{L}2$, e.g., the fragment $\Box\mu\mathcal{L}2$ without negation and the "possibility operator" $\Diamond$, by Proposition 24 and exploiting general results in [24], we can translate a temporal formula M over $\mathcal{G}$ where the atomic propositions are reflected by edge-bijective morphisms to a temporal formula $M_2[M]$ over markings (using for atomic propositions the encoding mentioned above), ensuring that if $\mathcal{C}^k(\mathcal{G}) \models M_2[M]$ then $\mathcal{G} \models M$, i.e., M is valid for the original GTS. We conclude by recalling that temporal state-based logics over Petri nets, i.e., logics where basic predicates have the form $\#s \leq c$, are not decidable in general, but important fragments of such logics are [20].

6 Conclusions

We presented an overview of the work on the unfolding semantics of GTSs, discussing its role for the development of a functorial concurrent semantics for GTSs and its possible applications to the verification of (infinite and finite state) systems modelled as GTSs. We used the SPO approach since due to the absence of the dangling condition it provides us with a more elegant unfolding semantics, but a large part of the theory can be equally developed for the DPO approach. For the approaches to verification, which deal with node-preserving grammars, the choice between SPO and DPO is immaterial.

The framework can appear fairly abstract and theoretical in nature. However, a prototype tool called AUGUR [21] has been implemented for computing the k-covering of a given graph transformation system. The current implementation is restricted to rules with discrete contexts. The tool can be downloaded at `http://www.ti.inf.uni-due.de/research/augur_1/`. The input and output of AUGUR is in GXL and GTXL, an XML standard for the exchange of graphs and graph transformation systems. Suitable converters have been added in order to visualize rules and Petri graphs and to extract the Petri net component of a Petri graph, which can then be analyzed with standard algorithms for nets.

Concerning the verification of finite state systems, the approach based on the construction of a finite complete prefix of the unfolding currently only applies to a special class of GTSs (read persistent GTSs). The problem of generalising the technique to the full class of GTSs is still open. An algorithm solving a similar problem in the simpler case of *contextual* Petri nets has been proposed recently [6] and we are confident that this can be adapted to GTSs.

A very stimulating direction of further research is the extension of the work on unfolding to the setting of rewriting systems over adhesive categories. Adhesive categories [22] have been recently introduced as an elegant and extremely general framework where the algebraic approaches to rewriting can be developed, encompassing rewriting on (several brands of) graphs and more general graphical structures like bigraphs or UML models. An unfolding theory for adhesive rewriting systems would thus apply uniformly to all these structures. Some promising steps have been taken in [3], which develops a concurrent semantics for adhesive rewriting systems based on deterministic processes.

References

1. P. Baldan. *Modelling concurrent computations: from contextual Petri nets to graph grammars.* PhD thesis, Department of Computer Science, University of Pisa, 2000. Available as technical report n. TD-1/00.
2. P. Baldan, A. Corradini, J. Esparza, T. Heindel, B. König, and V. Kozioura. Verifying Red-Black Trees. In *Proc. of COSMICAH 2005*, volume RR-05-04, pages 1–15. Queen Mary University, Dept. of Computer Science, 2005.
3. P. Baldan, A. Corradini, T. Heindel, B. König, and P. Sobociński. Processes for adhesive rewriting systems. In W. Aceto and A. Ingólfsdóttir, editors, *Proceedings of FoSSaCS '06*, volume 3921 of *LNCS*, pages 202–216. Springer, 2006.
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 2001*, volume 2154 of *LNCS*, pages 381–395. Springer, 2001.
5. 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 *LNCS*, pages 83–98. Springer, 2004. Full version as Tech. Rep. CS-2004-10, Dept. of Comp. Sci., University Ca' Foscari of Venice.
6. P. Baldan, A. Corradini, B. König, and S. Schwoon. McMillan's Complete Prefix for Contextual Nets. *ToPNoC - Trans. on Petri Nets and Other Models of Concurrency*, 2008. Special Issue from PN 2007 Workshops and Tutorials, to appear.
7. P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In *Proc. of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer, 1999.
8. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.
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 *Proc. of ICGT'02*, volume 2505 of *LNCS*, pages 14–29. Springer, 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*, volume 2694 of *LNCS*, pages 255–272. Springer, 2003.

12. R. Bruni, H.C. Melgratti, and U. Montanari. Event structure semantics for nominal calculi. In C. Baier and H. Hermanns, editors, *Proceedings of CONCUR 2006*, volume 4137 of *LNCS*, pages 295–309. Springer, 2006.
13. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.
14. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg [31].
15. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic Approaches to Graph Transformation II: Single Pushout Approach and comparison with Double Pushout Approach. In Rozenberg [31].
16. H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.
17. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, 1994.
18. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(20):285–310, 2002.
19. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
20. R.R. Howell, L.E. Rosier, and H. Yen. A taxonomy of fairness and temporal logic problems for Petri nets. *Theoretical Computer Science*, 82:341–372, 1991.
21. B. König and V. Kozioura. AUGUR 2—a new version of a tool for the analysis of graph transformation systems. In R. Bruni and D. Varró, editors, *Proceedings of GT-VMT '06 (Workshop on Graph Transformation and Visual Modeling Techniques)*, ENTCS. Elsevier, 2006. To appear.
22. S. Lack and P. Sobociński. Adhesive categories. In I. Walukiewicz, editor, *Proceedings of FoSSaCS'04*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.
23. R. Langerak. *Transformation and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.
24. 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.
25. K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
26. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In *Proceedings of CONCUR '92*, volume 630 of *LNCS*, pages 286–301. Springer, 1992.
27. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
28. G.M. Pinna and A. Poigné. On the nature of events: another perspective in concurrency. *Theoretical Computer Science*, 138(2):425–454, 1995.
29. W. Reisig. *Petri Nets: An Introduction*. EACTS Monographs on Theoretical Computer Science. Springer, 1985.
30. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.
31. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, 1997.
32. W. Vogler, A. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 501–516. Springer, 1998.
33. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.