A Static Analysis Technique for Graph Transformation Systems[⋆]

Paolo Baldan, Andrea Corradini, and Barbara König

Dipartimento di Informatica, Università di Pisa, Italia
{baldan,andrea,koenigb}@di.unipi.it

Abstract. In this paper we introduce a static analysis technique for graph transformation systems. We present an algorithm which, given a graph transformation system and a start graph, produces a finite structure consisting of a hypergraph decorated with transitions (Petri graph) which can be seen as an approximation of the Winskel style unfolding of the graph transformation system. The fact that any reachable graph has an homomorphic image in the Petri graph and the additional causal information provided by transitions allow us to prove several interesting properties of the original system. As an application of the proposed technique we show how it can be used to verify the absence of deadlocks in an infinite-state Dining Philosophers system.

1 Introduction

Graphs are very useful to describe complex structures in a direct and intuitive way. Graph Transformation Systems (GTSs) [18] add to the static description given by graphs a further dimension which models graph evolution via the application of rules, usually having local effects only. GTSs have been recognized to have fruitful applications in various fields of Computer Science [5], and specifically in the modelling and specification of concurrent and distributed systems [6].

As a high-level specification formalism for concurrent systems, GTSs are known to be more expressive than (Place/Transition) Petri nets, which can be seen, indeed, as GTSs acting on discrete graphs only (i.e., multisets of tokens) [3]. However, even if the theory of GTSs is nowadays well developed and a number of tools for the support of specifications based on this formalism have been developed, GTSs are not yet used as widely as Petri nets. One reason for this could be the lack of analysis techniques, which have been proven to be extremely effective for Petri nets. Verification and validation techniques play an important role during the design of the specification of a complex system, as they offer the designer the possibility to raise confidence in the quality of the specification, for example by allowing the early detection of logical errors.

While several static analysis techniques have been proposed for Petri nets, ranging from the calculus of invariants [16] to model checking based on finite

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

complete prefixes [13],[1] the rich literature on GTSs does not contain many contributions to the static analysis of such systems (see [11, 12]).

In this paper we present an original analysis technique for a class of (hyper)graph transformation systems, which, given a system and a start hypergraph, extracts from them an *approximated unfolding*, which is a finite structure (called *Petri graph*) consisting of a hypergraph and of a P/T net over it. Both the graphical and Petri net components of the approximated unfolding can be used to analyze the original system. For example, we will show that every hypergraph reachable from the start graph can be mapped homomorphically to the (graphical component of the) approximated unfolding. Therefore, if a property over graphs is reflected by graph morphisms, then if it holds on the approximated unfolding it also holds on all reachable graphs. Among these properties we mention the non-existence and non-adjacency of edges with specific labels, the absence of certain paths (for checking security properties) or cycles (for checking deadlock-freedom). Furthermore, the transitions of the Petri net component of the approximated unfolding can be seen as (approximated) occurrences of rules of the original graph transformation system, and indeed every reachable graph of the GTS corresponds (in a sense formalized later) to a reachable marking of the net. This allows one to prove other properties directly on the Petri net component, including upper and lower bounds on the number of times an edge with a certain label is present in a reachable graph and certain causal dependencies among rule applications. Notice that in general the net component of the approximated unfolding is neither safe nor acyclic; roughly one can say that, at least for certain properties, the analysis of a graph transformation system can be reduced to the analysis of a Petri net, which is a computationally less powerful model and for which the existing analysis techniques can be used.

The construction of the approximated unfolding of a graph transformation system is similar in spirit to the construction of the finite complete prefix [13] of a net, but more complex. Both are based on the unfolding construction, which in the case of nets [15] unwinds a Petri net into a branching occurrence net (a particularly simple Petri net satisfying suitable acyclicity and conflict freeness requirements), behaviourally equivalent to the original net. The unfolding cannot be used "directly" for verification purposes, since it is usually infinite. In the case of bounded nets, McMillan has observed in [13] that it is possible to truncate the unfolding in such a way that the resulting finite structure, the *finite complete prefix*, contains as much information as the unfolding itself, and can therefore be used for checking efficiently behavioural properties ([8, 9, 19]).

The unfolding construction has been generalized to graph transformation systems [17, 2, 1], and the technique we propose makes use of unfolding steps for generating the (finite) approximated unfolding, but the analogy with the finite prefix construction of nets ends here. In fact the GTSs we consider are not finite-state in general, hence, we must abandon the idea of finding a complete *finite* part of the unfolding, where every state reachable in the considered GTS has

[1] We use the term "static analysis" in a quite wide meaning, as for example it is used in the community of the Static Analysis Symposia.

an *isomorphic* image. Even if we relax the last requirement, by asking only that every reachable state has an *homomorphic* image in the constructed unfolding, since the states of the systems we consider are more structured (graphs *versus* multisets), it is not possible to rudely truncate the unfolding construction: at certain stages we have to merge parts of the unfolding already constructed. Because of this merging, the resulting structure is not acyclic (unlike the finite complete prefixes), and part of the information on the causality and concurrency of the system is lost. For what concerns state reachability, every state reachable in the original system is also reachable in the approximated unfolding, but we loose the converse implication (which instead holds for the finite complete prefix).

Technically, the algorithm that computes the approximated unfolding of a GTS is defined through two basic transformations, called *unfolding* and *folding* operations, which are applied as long as possible to the (Petri graph representing the) start graph of the system. Since both folding and unfolding are applied only if certain conditions are satisfied, the algorithm can be shown to terminate, a fact which guarantees that the resulting Petri graph is finite. Furthermore, although the proposed algorithm is non-deterministic, a local confluence property of the unfolding and folding transformations ensures that the approximated unfolding of a GTS is uniquely determined.

The paper is organized as follows. In Section 2 we introduce the class of GTSs on which our static analysis technique will be defined, as well as Petri graphs and some basic operations on them. The algorithm computing the approximated unfolding of a GTS is presented in Section 3, while Section 4 collects the main results about the algorithm, namely its termination, its confluence, and the fact that every reachable graph can be mapped to a reachable subgraph of the approximated unfolding. Section 5 illustrates the proposed method by applying it to the classical dining philosophers, both in a finite- and in an infinite-state variant. Section 6 concludes and hints at possible developments of the ideas presented in the paper.

2 Hypergraph rewriting, Petri nets and Petri graphs

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

2.1 Graph transformation systems

In the following, given a set A we denote by A^* the set of finite strings of elements of A. Furthermore, if $f : A \to B$ is a function then we denote by $f^* : A^* \to B^*$ its extension to strings. Throughout the paper Λ denotes a fixed set of *labels* and 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 finite set of nodes, E_G is a finite 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 ${\varphi_V}^*(c_G(e)) = c_{G'}(\varphi_E(e))$.*

In the sequel, when dealing with hypergraph morphisms we will often omit the subscripts V and E when referring to the components of a morphism φ.

Definition 2 (rewriting rule). *A* rewriting rule *r is a triple (L, R, α), where L and R are hypergraphs, called* left-hand side *and* right-hand side*, respectively, and $\alpha : V_L \to V_R$ is an injective mapping.*

A rule $r = (L, R, \alpha)$ is called basic *if l_L is injective, i.e., different edges in the left-hand side L have different labels, no node in L is isolated and no node in $V_R - \alpha(V_L)$ is isolated in R.*

In the following we will consider *only* basic rules. This restriction is not strictly needed, but makes the presentation simpler. For example, a morphism of a left-hand side into a hypergraph is completely determined by the image of its edges. Furthermore, to simplify the notation we will assume, without loss of generality, that $V_L \subseteq V_R$, $E_L \cap E_R = \emptyset$ and that the mapping α is the identity.

Intuitively, a rule $r = (L, R, \alpha)$ specifies that an occurrence of the left-hand side L can be "replaced" by R, according to the following definition.

Definition 3 (hypergraph rewriting). *Let $r = (L, R, \alpha)$ be a rewriting rule. A* match *of r in a hypergraph G is any morphism $\varphi : L \to G$. In this case we write $G \Rightarrow_{r,\varphi} H$ or simply $G \Rightarrow_r H$, where H is defined as follows: $V_H = V_G \uplus (V_R - V_L)$, $E_H = (E_G - \varphi(E_L)) \uplus E_R$, and if $\overline{\varphi} : V_R \to V_H$ is the obvious extension of φ then*

$$c_H(e) = \begin{cases} c_G(e) & \text{if } e \in E_G - \varphi(E_L) \\ \overline{\varphi}^*(c_R(e)) & \text{if } e \in E_R \end{cases}, \qquad l_H(e) = \begin{cases} l_G(e) & \text{if } e \in E_G - \varphi(E_L) \\ l_R(e) & \text{if } e \in E_R \end{cases}$$

Given a graph transformation system (GTS)*, i.e., a finite set of rules $\mathcal{R}$, we write $G \Rightarrow_{\mathcal{R}} H$ if $G \Rightarrow_r H$ for some $r \in \mathcal{R}$. Furthermore we will denote the transitive closure of $\Rightarrow_{\mathcal{R}}$ by $\Rightarrow_{\mathcal{R}}^*$. A GTS with a start graph $(\mathcal{R}, G_{\mathcal{R}})$ is called a* graph grammar*.*

The application of the rule r to G at the match φ first removes from G the image of the edges of L. Then the graph G is extended by adding the new nodes in R (i.e., the nodes in $V_R - V_L$) and the edges of R. Observe that the (images of) the nodes in L are "preserved", i.e., not affected by the rewriting step.

The reader which is familiar with the double-pushout (DPO) approach [4] to graph rewriting would have recognized that our rules (L, R, α) can be seen as DPO rules ($L \hookleftarrow V_L \overset{\alpha}{\hookrightarrow} R$) and that our notion of rewriting is equivalent to a DPO construction. Hence compared to general DPO rules $L \overset{\varphi_L}{\leftarrow} K \overset{\varphi_R}{\rightarrow} R$ we have that (i) K is discrete, i.e., it contains no edges, (ii) no two edges in the left-hand side L have the same label, (iii) the morphism φ_L is surjective on nodes, (iv) V_L and $V_R - \varphi_R(V_K)$ do not contain isolated nodes.

2.2 Petri nets

In this subsection we fix some basic notation for Petri nets [16, 14]. 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 $m \oplus m'' = m'$ and *difference*, which, in the same situation, is defined by $m' - m = m''$. Furthermore, for $m \in A^{\oplus}$ and $a \in A$ we write $a \in m$ for $a \leq m$. Often we will confuse a subset $X \subseteq A$ with the multiset $\bigoplus_{x \in X} x$.

Definition 4 (Petri net). *Let A be a finite set of action labels. An A-*labelled Petri net *is a tuple $N = (S, T, {}^\bullet(), ()^\bullet, p)$ where S is a set of places, T is a set of transitions, ${}^\bullet(), ()^\bullet : T \to S^{\oplus}$ assign to each transition its pre-set and post-set and $p : T \to A$ assigns an action label to each transition.*

The Petri net is called irredundant *if there are no distinct transitions with the same label and pre-set, i.e., if for any $t, t' \in T$*

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

A marked Petri net *is pair (N, m_N), where N is a Petri net and $m_N \in S^{\oplus}$ is the initial marking.*

The irredundancy condition (1) requires that two distinct transitions differ for the label or for the pre-set. This condition, in the case of branching processes, allows one to interpret each transition as an occurrence of firing of a transition in the original net, uniquely determined by its causal history (see [7]). Similarly, here it aims at avoiding the presence of multiple events which are indistinguishable for what regards the behaviour of the system. Hereafter *all* the considered Petri nets will be implicitly assumed irredundant, unless stated otherwise.

Definition 5 (causality relation). *Let N be a (marked) Petri net. The* causality relation *$<_N$ over N is the least transitive relation such that, for any $t \in T$, $s \in S$, we have (i) $s <_N t$ if $s \in {}^\bullet t$ and (ii) $t <_N s$ if $s \in t^\bullet$. For any $x \in S \cup T$ we define its sets of* causes *$\lfloor x \rfloor = \{y \in S \cup T \mid y <_N x\}$.*

Observe that, since we want to use Petri nets to represent the causality structure of a system only in an approximated way, no assumptions are made concerning the acyclicity of the net.

2.3 Petri graphs

We next introduce the structure that we intend to use to approximate graph transformation systems, the so-called Petri graphs, which consist of an hypergraph and of a Petri net whose places are the edges of the graph.

Definition 6 (Petri graph). *Let $\mathcal{R}$ be a GTS. A Petri graph (over $\mathcal{R}$) is a tuple $P = (G, N, \mu)$ where G is a hypergraph, $N = (E_G, T_N, {}^\bullet(), ()^\bullet, p_N)$ is an $\mathcal{R}$-labelled Petri net where the places are the edges of G, and μ associates to each transition $t \in T_N$, with $p_N(t) = (L, R, \alpha)$, a hypergraph morphism $\mu(t) : L \cup R \to G$ such that*

$$^\bullet t = \mu(t)^{\oplus}(E_L) \ \wedge \ t^\bullet = \mu(t)^{\oplus}(E_R) \tag{2}$$

A Petri graph for a graph grammar $(\mathcal{R}, G_{\mathcal{R}})$ is a pair (P, ι) where $P = (G, N, \mu)$ is a Petri graph for $\mathcal{R}$ and $\iota : G_{\mathcal{R}} \to G$ is a graph morphism. The multiset $\iota^{\oplus}(E_{G_{\mathcal{R}}})$ is called the initial marking *of the Petri graph. A marking $m \in {E_G}^{\oplus}$ will be called* reachable (coverable) *in (P, ι) if it is reachable (coverable) in the underlying Petri net.*

Condition (2) requires that each transition in the net can be viewed as an "occurrence" of a rule in $\mathcal{R}$. More precisely, if $p_N(t) = (L, R, \alpha)$ and $\mu(t) : L \cup R \to G$ is the morphism associated to the transition, then $\mu(t)_{|L} : L \to G$ must be a match of the rule in G such that the image of the edges of L in G coincides with the pre-set of t. Observe that, due to the assumption on the rules (no multiple labels and no isolated node in the left-hand side) the morphism $\mu(t)_{|L}$ (if it exists) is completely determined by $^\bullet t$. Then, the result of applying the rule to the considered match must be already in graph G, and the corresponding edges must coincide with the post-set of t. This is formalized by the condition over the image through $\mu(t)$ of the edges of R (note that the set E_R is seen as a multiset and $\mu(t)$ as a multiset function to take care of multiplicities).

Every hypergraph G can be considered as a Petri graph $[G] = (G, N, \mu)$ for $\mathcal{R}$, by taking N as the net with $S_N = E_G$ and no transitions. Similarly, $G_{\mathcal{R}}$ can be seen as Petri graph for $(\mathcal{R}, G_{\mathcal{R}})$ by taking as $\iota : G_{\mathcal{R}} \to G_{\mathcal{R}}$ the identity.

We now introduce a merging operation on Petri graphs which constructs the quotient of a Petri graph through an equivalence induced by a suitable relation.

Definition 7 (consistent and closed relation on a Petri graph). *Let $P = (G, N, \mu)$ be a Petri graph and let $\frown$ be a relation on $V_G \cup E_G \cup T_N$ (assume the sets V_G, E_G, T_N to be disjoint). We say that $\frown$ is* consistent *when (i) if $x \frown x'$ then $x, x' \in X$ for some $X \in \{V_G, E_G, T_N\}$, (ii) for all $e, e' \in E_G$ if $e \frown e'$ then $l_G(e) = l_G(e')$ and (iii) for all $t, t' \in T_N$, if $t \frown t'$ then $p_N(t) = p_N(t')$.*

A consistent relation $\frown$ over P is called closed *if for all $t, t' \in T_N$, $e, e' \in E_G$*

$$p_N(t) = p_N(t') = (L, R, \alpha) \ \wedge \ (\forall e \in E_L : \mu(t)(e) \frown \mu(t')(e)) \ \Rightarrow \ t \frown t' \tag{3}$$

$$t \frown t' \Rightarrow \ \forall e \in E_L \cup E_R : \mu(t)(e) \frown \mu(t')(e) \tag{4}$$

$$e \frown e' \ \wedge \ c_G(e) = v_1 \dots v_m \ \wedge \ c_G(e') = v'_1 \dots v'_m \ \Rightarrow \ \forall 1 \le i \le m : v_i \frown v'_i \tag{5}$$

To ensure that the quotient of a Petri graph with respect to a relation is well-defined and irredundant, the relation must be closed. Hence the simple observation below is essential for defining the merging operation.

Fact. Given any consistent relation $\frown$ over a Petri graph P there exists a least equivalence relation $\approx$ including $\frown$ and closed.

Definition 8 (Petri graph merging). *Let $P = (G, N, \mu)$ be a Petri graph and let $\frown$ be a consistent relation over P. Then the merging of P w.r.t. $\frown$, denoted by $P /\!\!/_{\frown}$, is the Petri graph (G', N', μ') defined as follows. Let $\approx$ be the least equivalence relation extending $\frown$ and closed in the sense of Definition 7. Then*

$$G' = (V_G/_{\approx}, E_G/_{\approx}, c_{G'}, l_{G'}),$$

where $c_{G'}([e]_{\approx}) = [v_1]_{\approx} \dots [v_n]_{\approx}$ and $l_{G'}([e]_{\approx}) = l_G(e)$ whenever $e \in E_G$ and $c_G(e) = v_1 \dots v_n$. Furthermore $N' = (E_{G'}, T_N/_{\approx}, {}^\bullet(), ()^\bullet, p_{N'})$, where ${}^\bullet[t]_{\approx} = \bigoplus_{e \in {}^\bullet t} [e]_{\approx}$, $[t]_{\approx}{}^\bullet = \bigoplus_{e \in t^\bullet} [e]_{\approx}$ and $p_{N'}([t]_{\approx}) = p_N(t)$ whenever $t \in T_N$. For each $t \in T_N$ the morphism $\mu'([t]_{\approx})$ is defined by $\mu([t]_{\approx})(x) = [\mu(t)(x)]_{\approx}$ for any graph item x in the rule $p_N(t)$.

Given a graph morphism $h : H \to G$ we will denote by $h /\!\!/_{\frown} : H \to G'$ the corresponding morphism, defined by $h /\!\!/_{\frown}(x) = [h(x)]_{\approx}$ for any $x \in V_H \cup E_H$.

The merging operation can be extended to sets of Petri graphs. Let $P_i = (G_i, N_i, \mu_i)$, with $i \in \{1, \dots, n\}$, be Petri graphs and assume that the sets V_{G_i}, E_{G_i}, T_{N_i} are pairwise disjoint. Then the componentwise union $P = P_1 \cup \dots \cup P_n$ is a Petri graph. A relation $\frown$ over $P_1, \dots, P_n$ is called consistent (closed) if it is a consistent (closed) relation over P. Given a consistent relation over $P_1, \dots, P_n$, we define the merging $\{P_1, \dots, P_n\} /\!\!/_{\frown} = P /\!\!/_{\frown}$.

3 Algorithm computing the approximated unfolding

In this section we describe an algorithm which computes the approximated unfolding of a graph grammar. Given a graph grammar, the algorithm produces a *finite* Petri graph such that every graph reachable in the grammar corresponds, in a sense formalized later, to a marking which is reachable in the Petri graph.

Let $(\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar. Its ordinary unfolding [17, 2] is constructed inductively beginning from the start graph and then applying at each step in all possible ways the rules, without deleting the left-hand side, and recording each occurrence of a rule and each new graph item generated in the rewriting process. As a result one obtains an acyclic branching graph grammar describing the behaviour of $(\mathcal{R}, G_{\mathcal{R}})$. In particular every reachable graph embeds in (a concurrent subgraph of) the graph produced by the unfolding construction.

The unfolding is usually infinite, also in the case of finite-state systems. Here, to ensure that the our algorithm produces a finite structure, we consider—besides the *unfolding rule*, which extends the graph by simulating the application of a rule without deleting its left-hand side—a *folding rule*, which allows us to "merge" two occurrences of the left-hand side of a rule whenever they are, in a sense made precise later, one causally dependent on the other.

Definition 9 (folding operation). *Let $P = (G, N, \mu)$ be a Petri graph for a GTS $\mathcal{R}$. Let $r = (L, R, \alpha) \in \mathcal{R}$ be a rule and let $\varphi', \varphi : L \to G$ be matches of r in G. Let $\frown$ be the relation over P defined as follows: for every $e \in E_L$*

$$\varphi'(e) \frown \varphi(e).$$

The folding *of P at the matches φ', φ is the Petri graph* $\mathsf{fold}(P, r, \varphi', \varphi) = P/\!\!/_{\frown}$. *If (P, ι) is a Petri graph for a graph grammar $(\mathcal{R}, G_{\mathcal{R}})$, in the same situation, we define* $\mathsf{fold}((P, \iota), r, \varphi', \varphi) = (P/\!\!/_{\frown}, \iota/\!\!/_{\frown})$.

To introduce the unfolding operation, we first need to fix some notation. If t is a transition and $r = (L, R, \alpha)$ is a rule we will write $P(t, r)$ to denote the Petri graph $(L \cup R, N, \mu)$ where $N = (E_{L\cup R}, \{t\}, {}^\bullet t = E_L, t^\bullet = E_R, p_N(t) = r)$ and $\mu(t) = id_{L\cup R}$. Whenever we can find a match of rule r in a given Petri graph, the unfolding operation extends the Petri graph by merging $P(t, r)$ at the match.

Definition 10 (unfolding operation). *Let $P = (G, N, \mu)$ be a Petri graph for a GTS $\mathcal{R}$. Let $r = (L, R, \alpha) \in \mathcal{R}$ be a rule and let $\varphi : L \to G$ be a match of r in G. Let $\frown$ be the relation over $\{P, P(t, r)\}$ defined as follows: for every $e \in E_L$*

$$\varphi(e) \frown e.$$

The unfolding *of P with rule r at match φ is the Petri graph* $\mathsf{unf}(P, r, \varphi) = \{P, P(t, r)\}/\!\!/_{\frown}$. *If (P, ι) is a Petri graph for a graph grammar $(\mathcal{R}, G_{\mathcal{R}})$, in the same situation, we define* $\mathsf{unf}((P, \iota), r, \varphi) = (\{P, P(t, r)\}/\!\!/_{\frown}, \iota/\!\!/_{\frown})$.

We can now describe the algorithm which produces the approximated unfolding of a given graph grammar. The algorithm generates a sequence of Petri graphs, beginning from the start graph and applying, non-deterministically, at each step, a folding or unfolding operation, until none of such steps is admitted.

Definition 11 (approximated unfolding). *Let $(\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar. The algorithm generates a sequence $(P_i, \iota_i)_{i\in\mathbb{N}}$ of Petri graphs as follows.*

(Step 0) *Initialize $(P_0, \iota_0) = ([G_{\mathcal{R}}], id_{G_{\mathcal{R}}})$.*

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

⋆ *Folding: Find a rule $r = (L, R, \alpha)$ in $\mathcal{R}$ and two matches $\varphi', \varphi : L \to G_i$ of r such that*

- *$\varphi^{\oplus}(E_L)$ is a coverable marking in P_i;*
- *there exists a transition $t \in T_{N_i}$ such that*

$$p_{N_i}(t) = r \;\wedge\; {}^\bullet t = \varphi'^{\oplus}(E_L) \;\wedge\; \forall e \in \varphi^{\oplus}(E_L) : (e \in {}^\bullet t \;\vee\; t <_{N_i} e). \quad (6)$$

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

⋆ *Unfolding: Find a rule $r = (L, R, \alpha)$ 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 ${}^\bullet t = \varphi^{\oplus}(E_L)$ and $p_{N_i}(t) = r$;*
- *there is no other match $\varphi' : L \to G_i$ satisfying condition (6).*

Then set $(P_{i+1}, \iota_{i+1}) = \mathsf{unf}((P_i, \iota_i), r, \varphi)$.

If no folding or unfolding step can be performed, the algorithm terminates. The resulting Petri graph (P_i, ι_i) is called the approximated unfolding *of $(\mathcal{R}, G_{\mathcal{R}})$ and denoted by $\mathcal{U}(\mathcal{R}, G_{\mathcal{R}})$.*

Condition (6) basically states that we can fold two matches 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 is already in the first one or causally depends on t. Roughly, the idea is that we should not unfold a left-hand side again, if we have already done the same unfolding step in its past, since this might lead to infinitely many steps. There are some similarities, to be further investigated, with the work in [10] where the sets of descendants and of normal forms of term rewriting systems are approximated by constructing an approximation automaton.

The coverability of a marking can be decided by computing the coverability tree of the net, as described in [16]. If this gets too costly, the condition of coverability can be relaxed or checked in an approximated way, a choice which does not compromise the result of correctness (see Proposition 12), but only reduces the "precision" of the algorithm: it will generate a worse approximation, where less properties of the given GTS can be proved.

4 Correctness, termination and confluence of the algorithm

We show that the algorithm described in the previous section is correct, namely that every reachable graph of a grammar is represented in the approximated unfolding produced by the algorithm. Furthermore the algorithm is terminating and confluent. Hence, by a classical result, its result is uniquely determined.

Correctness. We first show that the computed Petri graph is an appropriate approximation of the given graph grammar, in the sense that for any graph reachable in the graph grammar, there is a morphism into the approximated unfolding such that the image of its edge set corresponds to a reachable marking.

Proposition 12. *Let $(\mathcal{R}, G_{\mathcal{R}})$ be a graph grammar and assume that the algorithm computing the approximated unfolding terminates producing the Petri graph $\mathcal{U}(\mathcal{R}, G_{\mathcal{R}}) = ((U, N, \mu), \iota)$.*

Then for every graph G with $G_{\mathcal{R}} \Rightarrow_{\mathcal{R}}^{*} G$ there exists a morphism $\varphi_G : G \to U$ and the marking ${\varphi_G}^{\oplus}(E_G)$ is reachable in $\mathcal{U}(\mathcal{R}, G_{\mathcal{R}})$. Furthermore, if $G \Rightarrow_{\mathcal{R}} G'$ then ${\varphi_G}^{\oplus}(E_G) \xrightarrow{t} {\varphi_{G'}}^{\oplus}(E_{G'})$ for a suitable transition t in $\mathcal{U}(\mathcal{R}, G_{\mathcal{R}})$.

Termination. The basic result towards the proof of termination shows that it is not possible to perform infinitely many unfolding steps, without having the folding condition satisfied at some stage. This property is independent of the graph structure and can be proved by considering only the causality structure of a Petri graph, as expressed by the underlying Petri net. More formally, we show that in any infinite Petri net, satisfying suitable acyclicity and well-foundedness requirement, 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 (6) in Definition 11. Let us start formalizing the notion of folding pair.

Definition 13. *Let $N = (S, T, {}^\bullet(), ()^\bullet, p)$ be a Petri net. A* folding pair *in N is a pair of transitions $t, t' \in T$ such that $t \neq t'$, $p(t) = p(t')$ and for all $s \in {}^\bullet t'$ either $s \in {}^\bullet t$ or $t <_N s$.*

The next key lemma ensures that in any infinite net obtained by applying only unfolding steps there exists a folding pair.

Lemma 14. *Let $N = (S, T, {}^\bullet(), ()^\bullet, p)$ be an infinite irredundant Petri net, labelled over a finite set A, and satisfying the following conditions:*

- *for any $x \in S \cup T$ the set $\lfloor x \rfloor$ (the causes of x) is finite;*
- *the set $Min(N) = \{s \mid \lfloor s \rfloor = \emptyset\}$ is finite, i.e., only finitely many places have an empty set of causes;*
- *the relation $<_N$ is acyclic;*
- *the pre-set ${}^\bullet t$ of each transition is a set (rather than a proper multiset);*
- *for $t, t' \in T$ with $p(t) = p(t')$ it holds that $|{}^\bullet t| = |{}^\bullet t'|$.*

Then net N contains a folding pair.

Proof (Sketch). The core of the proof shows that if $Q \subseteq T$ is a set of transitions with the same action label a, then either there is a folding pair in Q or we can remove almost all elements of Q from N in a way that the resulting net remains infinite, i.e., there exists a set $Q' \subseteq Q$ such that $Q - Q'$ is finite and and the net obtained from N removing Q and all its causal consequences is infinite.

Then the result can be proved by induction on the number of labels that occur infinitely often in N. $\square$

The above lemma ensures that in our algorithm a folding step will be eventually performed. We have yet to show termination of the algorithm.

Proposition 15. *The algorithm computing the approximated unfolding (see Definition 11) terminates for every graph grammar $(\mathcal{R}, G_\mathcal{R})$.*

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. The following proposition only holds if we consider irredundant Petri nets.

Proposition 16. *Let us write $(P, \iota) \dashrightarrow (P', \iota')$ whenever (P, ι) can be transformed into (P'', ι'') by either a folding or an unfolding step applied under the corresponding condition (see the algorithm in Definition 11) and (P'', ι'') is isomorphic to (P', ι'), i.e., equal up to injective renaming of the edges, nodes and transitions.*

Let $(P, \iota) \dashrightarrow (P_i, \iota_i)$ for $i \in \{1, 2\}$. Then there is a Petri graph (P', ι') such that $(P_i, \iota_i) \dashrightarrow^* (P', \iota')$.

Since for a rewriting system local confluence and termination imply confluence we conclude the following result.

Proposition 17. *For any input $(\mathcal{R}, G_\mathcal{R})$ the algorithm computing the approximated unfolding terminates with a result $\mathcal{U}(\mathcal{R}, G_\mathcal{R})$ unique up to isomorphism.*

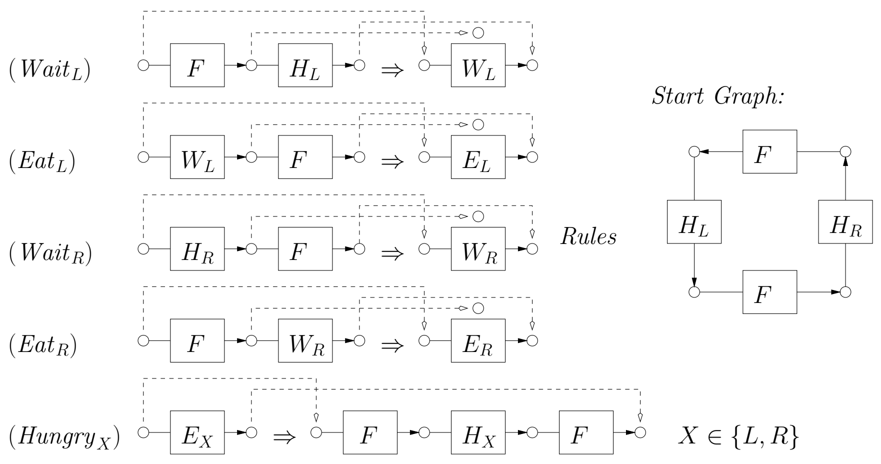


Fig. 1. A graph grammar modelling the dining philosophers (finite-state version).

5 The approximated unfolding at work: checking absence of deadlocks for dining philosophers

In order to illustrate our method, in this section we show how it can be applied to a well-known example, the dining philosophers system, which is presented in two versions, finite- and infinite-state.

Let us start with the classical finite-state version of the problem. Assume that sitting at the table are a left-handed philosopher and a right-handed philosopher with two forks between them. Our method is also applicable to instances of the problem with a greater number of philosophers. The restriction to two philosophers only avoids that the involved graphs become very large and hard to draw.

A philosopher, modelled by a binary edge, cycles through states H_X (hungry), W_X (waiting for the second fork), E_X (eating) where $X \in \{L, R\}$ depending on whether the philosopher is left- or right-handed. The thinking state is omitted. A fork is also represented by a binary edge labelled F. The system is described by the set of rewriting rules and by the start graph depicted in Fig. 1. A rule (L, R, α) is drawn in the form $L \Rightarrow R$, where edges are depicted by square boxes which are connected to a source node (the first node) and a target node (the second node). The mapping α is indicated by dashed arrows.

The algorithm in Definition 11 produces the Petri graph (a) in Fig. 2. Transitions are depicted by small rectangles and the connection to their pre-sets and post-sets is indicated by dashed arrows.

The algorithm terminates after six unfolding steps and four folding steps. Two unfolding steps which apply rules ($Wait_L$) and (Eat_L), respectively, to edge H_L with the corresponding forks, give rise to edge E_L. Then a further unfolding step using rule ($Hungry_L$) unfolds this edge into a graph consisting of two edges labelled F and one edge labelled H_L. But this graph consists of two

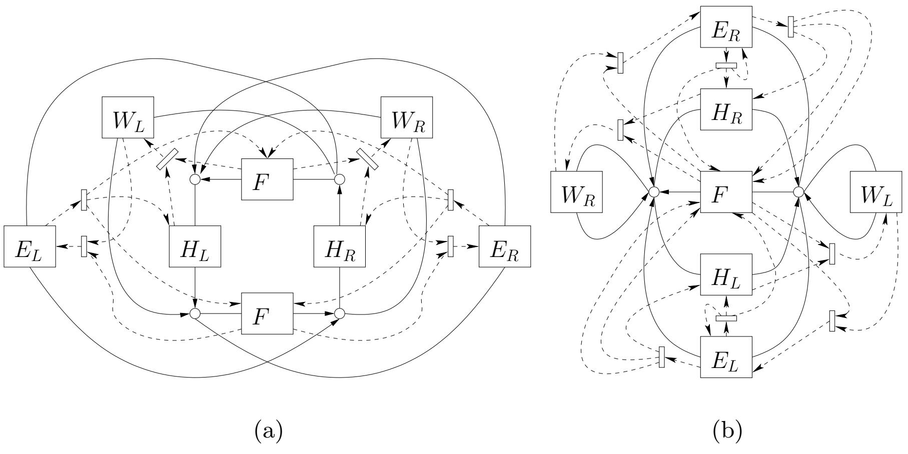


Fig. 2. Approximated unfoldings as Petri graphs: (a) dining philosophers, finite-state version; (b) dining philosophers, infinite-state version.

left-hand sides of previously applied rules and the edges are causally dependent on the corresponding transitions. Hence two folding steps can be applied, that merge the three edges (F, H_L and F) of the newly unfolded graph with the original edges from which they were derived. A symmetric reasoning applies for edge H_R.

We would like to prove that no deadlocks can occur in the system. First observe that any reachable graph is a cycle and, since an eating philosopher can always be reduced, a deadlocked state is necessarily a cycle including only hungry and waiting philosophers, where no forks are to the left of a left-handed hungry or a right-handed waiting philosopher and no forks are to the right of a right-handed hungry and a left-handed waiting philosopher. The absence of cycles is a property reflected by graph morphisms. Thus we can try to verify the absence of deadlocked states by analyzing cycles in the hypergraph associated to the approximated unfolding. To this aim we consider such graph as a finite-state automaton over the alphabet $\Sigma = \{F, H_X, W_X, E_X \mid X \in \{L, R\}\}$—with nodes as states and edges as transitions—and declare one of the four nodes as the initial and final state, thereby obtaining the languages L_{nw} (northwest node), L_{ne} (northeast node), L_{sw} (southwest node), L_{se} (southeast node). In this way we obtain all possible cycles of forks and philosophers as a regular language. By declaring, e.g., the northeast node as initial and final node we obtain the following language:

$$L_{ne} = (((FH_L + W_L)(E_RH_L)^*F + E_L)(W_RH_L(E_RH_L)^*F)^*H_R)^*.$$

An additional analysis of the Petri net would of course reveal that only a small finite subset of L_{ne} will ever occur, but here this is not needed for the analysis.

The language of all cycles allowing for the application of a rewrite rule is

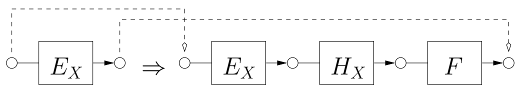
$$L_{lhs} = \Sigma^* E_L \Sigma^* + \Sigma^* E_R \Sigma^* + \Sigma^* F H_L \Sigma^* + H_L \Sigma^* F + \Sigma^* H_R F \Sigma^* + \\ F \Sigma^* H_R + \Sigma^* W_L F \Sigma^* + F \Sigma^* W_L + \Sigma^* F W_R \Sigma^* + W_R \Sigma^* F.$$

The language of all cycles which may occur but which do not allow the application of any rewriting rule can be now computed as $(L_{nw} \cup L_{ne} \cup L_{sw} \cup L_{se}) - L_{lhs} = \lambda$, i.e., the empty word. It is immediately clear that the circle of philosophers will never disappear entirely and thus we can conclude that no deadlocks will ever occur.

It is worth observing that if we forget about the graphical structure of the Petri graph, considering only the underlying Petri net, then we obtain a classical Petri net model of the dining philosophers. Therefore, in this case, the absence of deadlocks can be proved also by analyzing the Petri net underlying the approximated unfolding with classical Petri net techniques.

Now, in order to make things more interesting, we extend the example to an infinite-state system by adding a rule (Rep_X) which allows an eating philosopher to reproduce, creating another hungry philosopher with an adjacent fork.

(Rep_X)



$X \in \{L, R\}$

Observe that we can reuse the unfolding of the finite-state case and continue by unfolding the edges E_R and E_L using the two new rules. A sequence of further unfolding and folding steps causes the causes the two pairs of opposite nodes in the square to collapse, ending up with Petri graph (b) in Fig. 2.

Again we would like to prove that no deadlocks can occur. By declaring the left-hand node as initial and final state, we obtain the following language:

$$L_{left} = (W_R^*((H_L + H_R)W_L^*(F + E_L + E_R))^*)^*$$

while using the right-hand node in the same role, we obtain the language:

$$L_{right} = (W_L^*((F + E_L + E_R)W_R^*(H_L + H_R))^*)^*.$$

The language of all cycles which may occur but which do not allow the application of a rewriting rule can be now computed as $(L_{left} \cup L_{right}) - L_{lhs} = W_L^* + W_R^*$. Then, an analysis of the Petri net underlying the approximated unfolding reveals that actually no marking which consists of tokens exclusively in W_L or of tokens exclusively in W_R is reachable from the initial marking which consists of two tokens on F and one token on H_L and H_R each. Hence the system will never reach a deadlock.

Observe that in this case the analysis of the underlying Petri net by itself is not sufficient. In fact the Petri net can deadlock: we start from the initial marking and after the firing of two transitions we obtain a marking with one token on W_R and one token on W_L, where no further firing is possible.

6 Conclusion

We have presented a static analysis technique for graph transformation systems which produces a finite structure, called Petri graph, combining hypergraphs and Petri nets, which approximates the graphs which are reachable in the original grammar. Such a structure can be used to check safety properties, like the absence of deadlocks, in the original system.

An interesting question which has only been brushed in the paper, concerns the techniques which should be used to extract information from a computed Petri graph. It is certainly possible to reuse most of the well-established analysis techniques developed for Petri nets in the literature, such as coverability trees. However, as observed in the example, also the graphical structure underlying a Petri graph might play an essential role when establishing a property of the system. Since every graph reachable in the original grammar can be mapped to the approximated unfolding through a graph morphism, all properties which are reflected by graph morphisms can be checked on the approximated unfolding. We are currently investigating a syntactical characterization of such a class of properties. Other interesting issues are the use of methods from formal language theory (as hinted at in the example) and of model checking techniques.

Another question is the following: what can we do when we fail to prove a property? Obviously, it might still be the case that the considered property holds of the system, but this fact cannot be derived from the approximated unfolding where we have lost too much information by over-approximating. A partial solution could be to refine the description of the system, by computing a better approximation of the "complete" unfolding. This can be done by delaying folding steps and unfolding the Petri graph a bit further, "freezing" some parts of the approximated unfolding in order to avoid that a folding step leads to confuse them with other parts. A sequence of subsequently better approximations should converge to the whole, usually infinite, unfolding. In connection to this it would be interesting to determine which kind of systems can be "approximated" in an exact way—maybe by variations of the folding condition—one candidate being certainly Petri nets.

It is our aim to extend the proposed analysis technique to more general forms of graph rewriting, e.g., to the general double-pushout approach. In this case, since also edges might be preserved by a rewriting rule, the Petri net underlying a Petri graph cannot be simply an ordinary net, but it will be necessary to resort to contextual nets as in [1].

Acknowledgements: We are grateful to Javier Esparza for his insightful suggestions and to Alin Stefanescu and Stefan Schwoon who helped us with the finite-automaton tool used to compute the languages in Section 5. We are also grateful to anonymous referees for their valuable comments.

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, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.
3. A. Corradini. Concurrent Graph and Term Graph Rewriting. In U. Montanari and V. Sassone, editors, *Proceedings CONCUR'96*, volume 1119 of *LNCS*, pages 438–464. Springer Verlag, 1996.
4. 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 *LNCS*, pages 1–69. Springer Verlag, 1979.
5. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools.* World Scientific, 1999.
6. 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.
7. J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.
8. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, 1994.
9. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In T. Margaria and B. Steffen, editors, *Proc. of TACAS'96*, volume 1055 of *LNCS*, pages 87–106. Springer-Verlag, 1966.
10. T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In T. Nipkow, editor, *Proceedings 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, pages 151–165. Springer Verlag, 1998.
11. M. Koch. *Integration of Graph Transformation and Temporal Logic for the Specification of Distributed Systems.* PhD thesis, Technische Universität Berlin, 2000.
12. B. König. A general framework for types in graph rewriting. In *Proc. of FST TCS 2000*, volume 1974 of *LNCS*, pages 373–384. Springer-Verlag, 2000.
13. K.L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.
14. J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.
15. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
16. W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
17. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* PhD thesis, Technische Universität Berlin, 1996.
18. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations.* World Scientific, 1997.
19. W. Vogler, A. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 501–516. Springer-Verlag, 1998.