Electronic Communications of the EASST
Volume 2 (2006)

Proceedings of the
Workshop on Petri Nets and Graph Transformation
(PNGT 2006)

Graph transformation systems, Petri nets and Semilinear Sets: Checking for the Absence of Forbidden Paths in Graphs

Barbara König

11 pages

Guest Editors: Paolo Baldan, Hartmut Ehrig, Julia Padberg, Grzegorz Rozenberg
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Graph transformation systems, Petri nets and Semilinear Sets: Checking for the Absence of Forbidden Paths in Graphs

Barbara König[1]

[1] barbara_koenig@uni-due.de, http://www.ti.inf.uni-due.de/people/koenig/
Abteilung für Informatik und Angewandte Kognitionswissenschaft
Universität Duisburg-Essen, Germany

Abstract: We introduce an analysis method that checks for the absence of (Euler) paths or cycles in the set of graphs reachable from a start graph via graph transformation rules. This technique is based on the approximation of graph transformation systems by Petri nets and on semilinear sets of markings. An important application is deadlock analysis in distributed systems.

Keywords: graph transformation systems, Petri nets, static analysis, verification, deadlock detection

1 Introduction.

Graph transformation systems [17] are a powerful specification formalism that can be used to model systems with a dynamic evolution, such as mobile and distributed systems. Other typical examples are evolving object graphs or the heap of a program with its pointer structures. The power to model features such as infinite state spaces, dynamic creation and deletion of objects and variable topologies leads to great challenges concerning analysis and verification techniques for such systems.

In recent years there have been several approaches directed specifically at the verification of graph transformation systems, for instance [15, 7, 19]. In this paper we continue to develop our approach [2, 4], which over-approximates graph transformation systems by Petri nets and allows to use existing techniques in order to analyze the resulting net.

An important issue in this technique is to translate properties of graphs and graph transformation systems into properties concerning Petri net markings and reachability of markings. We have so far studied how to handle the following properties:

- **Non-reachability of certain subgraphs:** given a fixed graph G, show that no reachable graph in the graph transformation system contains G as a subgraph. This can be done via coverability checking on the net, by either using coverability graphs [14, 11] or techniques for well-structured transition systems [9, 1, 10].

- **First-order or second-order monadic logic on graphs [5]:** given an over-approximating Petri net, formulas on graphs can be encoded into propositional formulas on markings using propositions of the form $\#s \leq c$ (the number of tokens in place s is always smaller than or equal to some constant c).

  We have also studied how to interleave first-order quantification with temporal operators and how to translate these formulas into a temporal logic on Petri nets [3].

The aim of this paper is to use deeper results from Petri net theory than the ones employed so far in order to check for the absence of Euler paths or cycles in graphs, i.e., paths which traverse every edge of the graph exactly once. For this we need the notion of semilinear sets of markings of Petri nets and techniques for (approximative) reachability checking [8] in this setting. An important application is to verify the absence of deadlocks, which often manifest themselves as cycles. Hence we will study an infinite-state version of the dining philosophers (which first appeared in [2]) as a running example.

2 Graph Transformation Systems and Petri nets

In this section we describe the notions of graph transformation system (GTS) and Petri net. Here we will use only directed graphs, but we also plan to generalize the methods presented here to hypergraphs.

Definition 1 (graphs and graph morphisms) Let Λ be a set of labels. A *graph* G is a tuple $(V_G, E_G, s_G, t_G, l_G)$, where V_G is a set of nodes, E_G is a set of edges, $s_G, t_G : E_G \to V_G$ are the source and target functions and $l_G : E_G \to \Lambda$ is the labeling function.

Let G_1 and G_2 be two graphs. A *graph morphism* (or simply *morphism*) $\varphi : G_1 \to G_2$ consists of a pair of total functions $\varphi_V : V_{G_1} \to V_{G_2}$ and $\varphi_E : E_{G_1} \to E_{G_2}$ such that for every $e \in E_{G_1}$ it holds that $l_{G_1}(e) = l_{G_2}(\varphi_E(e))$, $\varphi_V(s_{G_1}(e)) = s_{G_2}(\varphi_E(e))$ and $\varphi_V(t_{G_1}(e)) = t_{G_2}(\varphi_E(e))$. A morphism is called *edge-bijective (edge-injective)* whenever it is bijective (injective) on edges. It is an *isomorphism* whenever it is bijective on nodes and edges.

Graphs can be rewritten using rules of the following kind. Since there is a tradeoff between the complexity of rules and the power of the verification technique, we choose a very simple rule format.

Definition 2 (rewriting rules and steps) A *rewriting rule* r is a span of injective graph morphisms $L \overset{\varphi_L}{\leftarrow} K \overset{\varphi_R}{\rightarrow} R$ where (i) $E_L \neq \emptyset$, (ii) K is discrete, (iii) φ_L is bijective on nodes, (iv) there are no isolated nodes in L and each isolated node in R belongs to $\varphi_R(V_K)$.

A *match* of L in a graph G is given by an edge-injective graph morphism $\psi : L \to G$.

We say that a graph G can *evolve* to H (written: $G \Rightarrow_r H$) if we can obtain H from G by applying rule r to a match ψ as specified by the double-pushout approach to graph rewriting [6].

In the following we will assume that the morphisms φ_L, φ_R are inclusions and hence the union $L \cup R$ of the left-hand and right-hand sides is well-defined. Alternatively $L \cup R$ can be obtained as the pushout of φ_L and φ_R.

A *graph transformation system (GTS)* $\mathcal{G} = (\mathcal{R}, G_0)$ consists of a finite set of rewriting rules $\mathcal{R}$ together with an initial graph G_0.

Example 1 *As a running example we will use an infinite-state dining philosophers system (see Figure 1). In order to avoid deadlocks we mix left-handed and right-handed philosophers. There is a left-handed philosopher which can be in states H_L (hungry), W_L (waiting for the second fork) and E_L (eating); similarly for the right-handed philosopher. The difference between the two is*

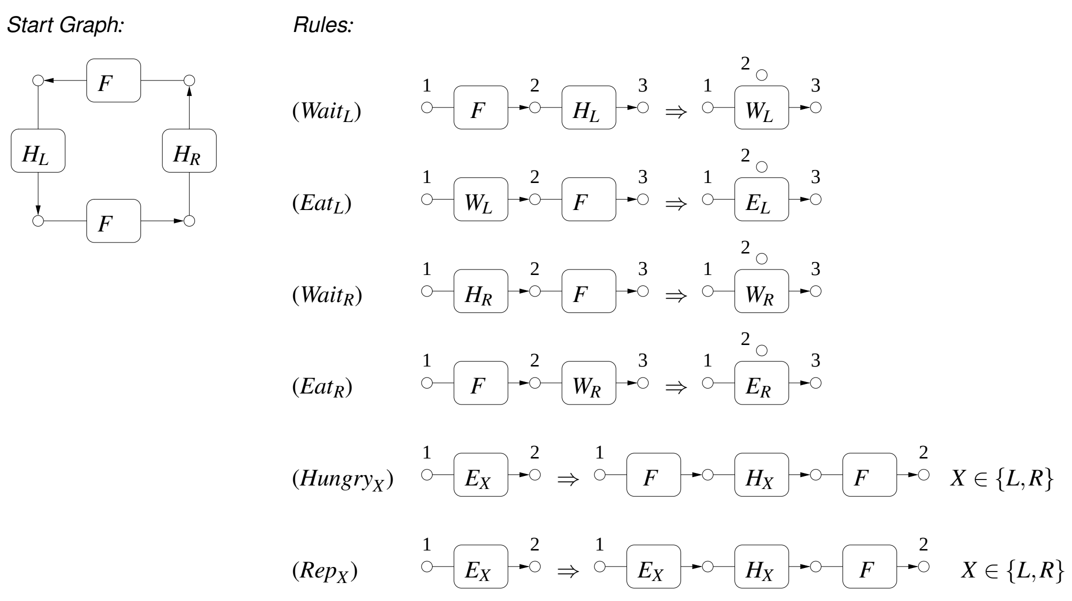

Start Graph:
F
H_L
H_R
F
Rules:
$(Wait_L)$
1 F 2 H_L 3 ⇒ 1 W_L 2 3
(Eat_L)
1 W_L 2 F 3 ⇒ 1 E_L 2 3
$(Wait_R)$
1 H_R 2 F 3 ⇒ 1 W_R 2 3
(Eat_R)
1 F 2 W_R 3 ⇒ 1 E_R 2 3
$(Hungry_X)$
1 E_X 2 ⇒ 1 F H_X F 2
$X \in \{L,R\}$
(Rep_X)
1 E_X 2 ⇒ 1 E_X H_X F 2
$X \in \{L,R\}$

Figure 1: Dining philosophers system represented by a GTS.

that the left-handed philosopher first picks up the fork to his left (rule $Wait_L$) and then the fork to his right (rule Eat_L), whereas the the right-handed philosopher behaves in the opposite way (rules $Wait_R$, Eat_R). Finally, both forks can be returned and the philosopher becomes hungry again (rule $Hungry_X$).

In order to obtain an infinite-state system we add a rule for replication which allows an eating philosopher to reproduce and create another hungry philosopher and a fork (rule Rep_X), thus increasing the number of philosophers at the table by one.

In order to approximate GTSs we will employ Petri nets, which, as multiset rewriting systems, can be seen as a special case of graph rewriting. By approximating with Petri nets we will be able to preserve nice properties of the GTS model, such as locality (state changes are only described locally) and concurrency (no unnecessary interleaving of events) in the approximation.

Some basic notation concerning *multisets* (or *markings*) is needed in order to deal with Petri nets. Given a set A we will denote by $A^\oplus$ the free commutative monoid over A, whose elements will be called *multisets* over A. The monoid operation is denoted by $\oplus$ and for $m \in A^\oplus$, $k \in \mathbb{N}_0$ we write $k \cdot m$ for $m \oplus \cdots \oplus m$ (k times). A multiset m can also be written as $\bigoplus_{a \in A} m(a) \cdot a$.

Given a function $f\colon A \to B$, by $f^\oplus\colon A^\oplus \to B^\oplus$ we denote its extension to multisets, i.e., $f^\oplus(m)(b) = \sum_{f(a)=b} m(a)$ for every $b \in B$.

Definition 3 (Petri net) Let Δ be a finite set of labels. A Δ-*labelled Petri net* is a tuple $N = (S, T, {}^\bullet(), ()^\bullet, p)$, where S is the set of places, T is a set of transitions, ${}^\bullet(), ()^\bullet\colon T \to S^\oplus$ assign to each transition its pre-set and post-set and $p : T \to \Delta$ assigns a label to each transition. A marked Petri net is a pair (N, m_N), where N is a Petri net and $m_N \in S^\oplus$ is the initial marking.

Given a set $\Theta \subseteq S$ of places we denote by ${}^\bullet\Theta$ the set of all transitions having a place of Θ in their post-sets and by $\Theta^\bullet$ the set of all transitions having a place of Θ in their pre-sets.

3 Approximated Unfolding

In this section we will give a short overview over the approximated unfolding technique described in [2]. First we define the notion of Petri graph which will be used to represent an over-approximation of a given GTS. Note that the edges of the graph are at the same time the places of the net and that the transitions are labelled with rules of the GTS.

Definition 4 (Petri graph) Let $\mathscr{G} = (\mathscr{R}, G_0)$ be a GTS. A *Petri graph* (over $\mathscr{R}$) is a tuple $P = (G, N, \mu)$, where G is a graph, $N = (E_G, T_N, {}^\bullet(), ()^\bullet, p_N)$ is an $\mathscr{R}$-labelled Petri net where the places are the edges of G and μ associates to each transition $t \in T_N$, with $p_N(t) = (L, R, id)$, a graph morphism $\mu(t) : L \cup R \to G$ such that ${}^\bullet t = \mu(t)^\oplus(E_L)$ and $t^\bullet = \mu(t)^\oplus(E_R)$.

A *Petri graph for the* GTS $\mathscr{G}$ is a pair (P, ι), where $P = (G, N, \mu)$ is a Petri graph over $\mathscr{R}$ and $\iota : G_0 \to G$ is a graph morphism. A marking is reachable (coverable) in Petri graph if it is reachable (coverable) in the underlying Petri net with the multiset $\iota^\oplus(E_{G_0})$ as the initial marking.

We view Petri graphs as symbolic representations of transition systems with graphs as states. Specifically each marking $m \in E_G^\oplus$ of a Petri graph (G, N, μ) can be seen as representation of a graph, denoted by *graph*(m), according to the following definition: We take the marked subgraph of G and duplicate (or multiply) each edge as indicated by the marking. (See also Example 3.)

More formally one can define *graph*(m) as the unique graph H, up to isomorphism, such that H has no isolated nodes and there exists a morphism $\psi : H \to G$, injective on nodes, with $\psi^\oplus(E_H) = m$.

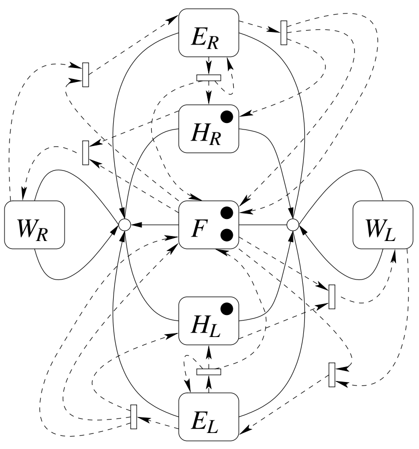
In order to obtain a Petri graph approximating a GTS, we first need—as building blocks—Petri graphs that describe the effect of a single rule.

Definition 5 (Petri graph for a rewriting rule) Let $r = (L, R, id)$ be a rewriting rule. By $P(t, r) = (G, N, \mu)$ we denote a Petri graph with $G = L \cup R$ and N is a net with places $S_N = E_L \cup E_R$ and one transition t such that $p_N(t) = r$, ${}^\bullet t = E_L$ and $t^\bullet = E_R$. Furthermore the morphism $\mu(t) \colon L \cup R \to G$ is the identity.

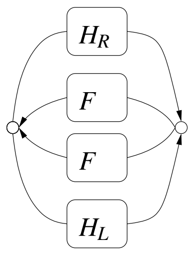
Given a GTS $\mathscr{G} = (\mathscr{R}, G_0)$ one can construct an over-approximating Petri graph $\mathscr{C}_\mathscr{G}$ (also called the *covering* of $\mathscr{G}$), using the following algorithm. It starts with a Petri graph P_0 that consists only of the start graph and computes $\mathscr{C}_\mathscr{G}$ iteratively. It is based on an unfolding technique which is combined with over-approximating folding steps which guarantee a finite approximation. Note that the covering returned is the coarsest over-approximation and there are various ways to refine it [4, 12].

Algorithm 1 (approximated unfolding) We set $P_0 = (G_0, N_0, \mu)$, where N_0 contains no transitions, $m_0 = E_{G_0}$ and let $\iota_0 \colon G_0 \to G_0$ be the identity. As long as one of the following steps is applicable, transform P_i into P_{i+1} according to one of the possibilities given below (where folding steps take precedence over unfolding steps).

Unfolding: Find a rule $r = (L, R, id) \in \mathscr{R}$ and a match $\varphi : L \to G_i$ such that $\varphi^\oplus(E_L)$ is coverable.



(a) Petri graph



(b) Graph $graph(m_N)$ generated by the initial marking m_N

Figure 2: Petri graph and generated graph for the dining philosophers example

Then choose a new transition t and extend P_i by attaching $P(t,r)$, i.e., take the disjoint union of both Petri graphs and factor through the equivalence $\equiv$ generated by $e \equiv \varphi(e)$ for every $e \in E_L$.

Folding: Find a rule $r = (L,R,id) \in \mathscr{R}$ and two matches $\varphi, \varphi' : L \to G_i$ such that $\varphi^{\oplus}(E_L)$ and $\varphi'^{\oplus}(E_L)$ are coverable in N_i and the second match is causally dependent on the transition unfolding the first match. Then merge the two matches by setting $\varphi(e) \equiv \varphi'(e)$ for each $e \in E_L$ and factoring through the resulting equivalence relation $\equiv$.

If neither possibility applies the Petri graph P_i obtained in the last step is returned. The result is denoted by $\mathscr{C}_{\mathscr{G}}$. In [2] it has been shown that the algorithm always terminates with a result unique up to isomorphism.

This approximated unfolding procedure has been implemented in the verification tool AUGUR[1], together with support for counterexample-guided abstraction refinement [12].

Example 2 *For our running example (dining philosophers), the algorithm above returns the Petri graph depicted in Figure 2(a). While the graph component is drawn as before, the Petri net transitions are depicted by small rectangular boxes with dashed lines connecting them to their pre- and post-sets. Furthermore the initial marking, i.e., the multiset image of the start graph under ι, is indicated.*

The algorithm above can be shown to be correct in the following sense.

Proposition 1 (correctness of the approximated unfolding algorithm [5]) *Let $\mathscr{G} = (\mathscr{R}, G_0)$ be a* GTS *and let $\mathscr{C}_{\mathscr{G}} = ((G,N,\mu),\iota)$ be the covering obtained by Algorithm 1. Then for every graph G that is reachable in $\mathscr{G}$ from G_0, there exists a marking m, reachable from the initial marking $\iota^{\oplus}(E_{G_0})$ in the net N, such that there is an edge-bijective morphism $G \to graph(m)$.*

[1] `http://www.ti.inf.uni-due.de/research/augur_1/`

Example 3 *The initial marking m_N of the Petri graph in Figure 2(a) generates the graph $graph(m_N)$ depicted in Figure 2(b). Note that the two tokens in the edge (place) labelled F result in two parallel F-edges. All edges that are unmarked by m_N disappear in $graph(m_N)$. Also note that there is an edge-bijective morphism from the start graph shown in Figure 1 to $graph(m_N)$. This morphism merges the north-west and the south-east node as well as the north-east and the south-west node of the initial graph.*

We will in the following show how to use the information provided by the over-approximation in order to verify that certain Euler paths or cycles do not appear in any reachable graph.

4 Euler Cycles and Semilinear Sets

Given a graph transformation system and a regular expression r, we now want to show that no reachable graph admits a (Euler) path such that the sequence of labels on this path is described by r. That is, the regular expression specifies a set of forbidden paths.

Definition 6 (Euler paths and cycles) Let G be a graph and r a regular expression over the alphabet Λ. Then we say that G contains a *path corresponding to r* if there is a sequence $e_1 \dots e_n$ of edges such that $t_G(e_i) = s_G(e_{i+1})$ for $i \in \{1, \dots, n-1\}$ and the word $l_G(e_1) \dots l_G(e_n)$ is contained in the language of r. This path is called an *Euler path* if every edge of G occurs in the sequence exactly once. The graph G contains an *Euler cycle corresponding to r* if it contains an Euler path for which additionally $t_G(e_n) = s_G(e_1)$ is satisfied.

As a starting point for our method we assume that the given (hyper-)graph transformation system $\mathcal{G}$ has already been approximated by a Petri graph $P = (G, N, \mu)$.

Now, given a regular expression r, the idea is to specify a set of markings $S \subseteq E_G^{\oplus}$ of the Petri graph P such that a marking m is contained in S if and only if $graph(m)$ has an Euler path (resp. cycle) corresponding to r. Since furthermore the property of having an Euler path is preserved by edge-bijective graph morphisms, it is sufficient to check that no such marking of S is reachable in the approximating net.

It will turn out that the set S is always *semilinar* [16], where semilinearity is defined as follows.

Definition 7 (semilinear sets) Let $S \subseteq E_G^{\oplus}$ be a set of markings. It is called *semilinear* if it is the finite union of sets of the form $\{m_0 \oplus k_1 \cdot m_1 \oplus \dots \oplus k_n \cdot m_n \mid k_i \in \mathbb{N}_0\}$, where $m_0, m_1, \dots, m_n$ are markings.

Furthermore S can be computed via products of finite automata as shown in the following proposition.

Proposition 2 (computing semilinear sets) *Let G be a graph and let r be a regular expression. Then there (constructively) exists a semilinear set $S_{G,r}^p$ such that for every $m \in E_G^{\oplus}$*

$$m \in S_{G,r}^p \iff graph(m) \text{ has an Euler path corresponding to } r.$$

Similarly, such a set $S_{G,r}^c$ can also be found for Euler cycles.

Proof. For Euler paths, first construct an automaton $\mathscr{A} = (Q_\mathscr{A}, \Lambda, \delta_\mathscr{A}, Q^0_\mathscr{A}, F_\mathscr{A})$ accepting the language of r, where $Q_\mathscr{A}$ is the set of states, Λ is the alphabet, $\delta_\mathscr{A}\colon Q_\mathscr{A} \times \Lambda \to \mathscr{P}(Q_\mathscr{A})$ is the transition function and $Q^0_\mathscr{A}, F_\mathscr{A} \subseteq Q_\mathscr{A}$ are the sets of initial respectively final states.

Now compute the cross product of G and $\mathscr{A}$ where the transitions of the resulting automaton $\mathscr{B}$ are labelled with the edge names of G. In more detail:

$$\begin{aligned} Q_\mathscr{B} &= V_G \times Q_\mathscr{A} \\ \Sigma_\mathscr{B} &= E_G \\ \delta_\mathscr{B}((v,q),e) &= \{(v',q') \mid s_G(e) = v, t_G(e) = v', q' \in \delta_\mathscr{A}(q, l_G(e))\} \\ Q^0_\mathscr{B} &= V_G \times Q^0_\mathscr{A} \\ F_\mathscr{B} &= V_G \times F_\mathscr{A} \end{aligned}$$

The automaton $\mathscr{B}$ accepts the set of paths of G that correspond to r. We consider the monoid morphism $\alpha\colon E^*_G \to E^\oplus_G$ from the free monoid $E^*_G = \Sigma^*_\mathscr{B}$ to the free commutative monoid $E^\oplus_G$ induced by $e \mapsto e$. Since $L(\mathscr{B})$ (the language generated by $\mathscr{B}$) is a rational set of $\Sigma^*_\mathscr{B}$ it holds that its image $S^p_{G,r} = \alpha(L(\mathscr{B}))$ is a rational and hence semilinear set of $E^\oplus_G$.

The easiest way to construct $S^p_{G,r}$ is to transform $\mathscr{B}$ into a regular expression r' and then to inductively apply a function α' converting r' to a semilinear set:

$$\begin{aligned} \alpha'(\emptyset) &= \emptyset \\ \alpha'(\varepsilon) &= \{0\} && \text{(where 0 is the empty multiset)} \\ \alpha'(e) &= 1 \cdot e && \text{(if } e \in E_G\text{)} \\ \alpha'(r_1 r_2) &= \{m_1 \oplus m_2 \mid m_1 \in \alpha'(r_1), m_2 \in \alpha'(r_2)\} \\ \alpha'(r_1 + r_2) &= \alpha'(r_1) \cup \alpha'(r_2) \\ \alpha'(r^*) &= \{k_1 \cdot m_1 \oplus \cdots \oplus k_n \cdot m_n \mid m_1, \dots, m_n \in \alpha'(r), k_1, \dots, k_n \in \mathbb{N}\} \end{aligned}$$

With some simplifications one then obtains a semilinear set as in Definition 7.

It remains to check that a marking m is contained in $S^p_{G,r}$ if and only if the graph generated by m allows an Euler path corresponding to r:

$$\begin{aligned} & m \in S^p_{G,r} = \alpha(L(\mathscr{B})) \\ \iff\ & \exists e_1 \dots e_n \in L(\mathscr{B}) \text{ s.t. } m = \alpha(e_1 \dots e_n) \\ \iff\ & e_1 \dots e_n \text{ is a path in graph } G \text{ which corresponds to } r \text{ and each edge } e \in E_G \text{ is} \\ & \quad \text{visited } m(e) \text{ times} \\ \iff\ & e_1 \dots e_n \text{ is a Euler path of } graph(m) \end{aligned}$$

For Euler cycles take several automata, one for each node $v \in V_G$. For each automaton let $\{v\} \times Q^0_\mathscr{A}$ be the set of initial states and $\{v\} \times F_\mathscr{A}$ the set of final states. This ensures that the accepted path starts and ends in the same node. Then take as $S^c_{G,r}$ the union of all languages generated by such automata. □

What is basically done in the construction above is to characterize the set of paths in G that correspond to r and then to enforce commutativity by forgetting about the ordering of edges on a path. This tells us how many times each edge should be present in order to allow an Euler path.

Note that it is not particularly surprising that we obtain a semilinear set in this way. In a sense this can be seen as a corollary of Parikh's theorem [13] which states that the Parikh image of every context-free language is semilinear. The Parikh image of a word language over the alphabet Σ is a set of multisets over Σ, where the order of alphabet symbols is forgotten and only their multiplicity is remembered. For instance, the Parikh image of the language $\{aabca, bbc, \dots\}$ is the set $\{3 \cdot a \oplus 1 \cdot b \oplus 1 \cdot c, 2 \cdot b \oplus 1 \cdot c, \dots\}$.

The function α' used in the proof of Proposition 2 converting regular expressions into semilinear sets works similarly: For instance let $r' = ab(b+c)^*ab$ be a regular expression. Then $\alpha'(r') = \{(2 \cdot a \oplus 2 \cdot b) \oplus k_1 \cdot b \oplus k_2 \cdot c \mid k_1, k_2 \in \mathbb{N}\}$.

Example 4 *We will now exemplify how semilinear sets can be used to verify the absence of deadlocks in our running example: by inspection of the rules we can observe that all reachable states are cycles and that furthermore the language of all cycles allowing a rule application is described by the following regular expression r_{lhs}:*

$$\Sigma^* E_L \Sigma^* + \Sigma^* E_R \Sigma^* + \Sigma^* F H_L \Sigma^* + H_L \Sigma^* F + \Sigma^* H_R F \Sigma^* +$$
$$F \Sigma^* H_R + \Sigma^* W_L F \Sigma^* + F \Sigma^* W_L + \Sigma^* F W_R \Sigma^* + W_R \Sigma^* F.$$

This regular expression can be obtained by taking the labels of the left-hand sides in Figure 1, ordering them as indicated by the edge directions and adding Σ^* in order to indicate other, unspecified, labels on the cycle.

Then the regular expression r describing forbidden deadlock states, allowing no rule application, can be obtained as the complement of r_{lhs}. (We do not give r explicitly, since it is fairly complex.)

Using the technique explained above we obtain the following semilinear set S for the Petri graph $\mathscr{C}_\mathscr{G}$ approximating the dining philosophers system (see Figure 2(a)):

$$S = \{k \cdot e(W_L) \mid k \in \mathbb{N}\} \cup \{k \cdot e(W_R) \mid k \in \mathbb{N}\}$$

where $e(W_L)$ and $e(W_R)$ are the only edges in $\mathscr{C}_\mathscr{G}$ labelled W_L and W_R respectively. This means, the only way to obtain an Euler cycle allowing no rule applications is to have a cycle consisting exclusively of W_L-edges or exclusively of W_R-edges.

It now remains to show that no marking in the set S is reachable in the Petri graph.

5 Analysis of Graph Transformation Systems.

Now if we could show that in the Petri graph no marking contained in $S^p_{G,r}$ ($S^c_{G,r}$) is reachable, we would have successfully verified that no reachable graph G contains a Euler path (cycle) corresponding to S. For if this were the case, there would be a reachable marking m and an edge-bijective graph morphism $G \to graph(m)$. Then, since edge-bijective morphisms preserve Euler cycles, there would be such a cycle in $graph(m)$, a contradiction to Proposition 2. This can be summarized in the following theorem.

Theorem 1 *Let $\mathcal{G} = (\mathcal{R}, G_0)$ be a* GTS *and let $\mathcal{C}_\mathcal{G} = ((G,N,\mu),\iota)$ be the covering obtained by Algorithm 1. Furthermore let r be a regular expression and let $S^p_{G,r}$, $S^c_{G,r}$ be the semilinear sets of Proposition 2 (for paths respectively cycles).*

If no marking of N, reachable from $\iota^\oplus(E_{G_0})$, is contained in $S^p_{G,r}$, then no reachable graph of $\mathcal{G}$ has a path corresponding to r. Furthermore, if no marking of N, reachable from $\iota^\oplus(E_{G_0})$, is contained in $S^c_{G,r}$, then no reachable graph of $\mathcal{G}$ has a cycle corresponding to r.

The problem of checking whether a marking of a semilinear set S is reachable in a net N is decidable, since it is reducible to the reachability problem for Petri nets [16]. However, since this problem is not known to be primitive recursive and all known algorithms are very complex, it will usually be necessary to resort to approximative reachability methods based on the marking equation and traps [8, 18, 20]. In the following we will study how traps can be used for approximative reachability checks.

Definition 8 (trap) Let N be a Petri net. A *trap* is a set Θ of places such that $\Theta^\bullet \subseteq {}^\bullet\Theta$, i.e., every transition that removes a token from a trap must also put a token into the trap.

For a trap Θ it always holds that once it is marked, it will always be marked. Hence, if Θ is initially marked, then a marking m which does not have a token inside Θ can not be reachable.

Example 5 *In the dining philosophers system the set $\Theta = \{e(F), e(W_L), e(E_L)\}$ is a trap. Hence we can conclude that a marking consisting only of tokens in W_R can not be reached. Symetrically one can show that a marking consisting only of tokens in W_L is not reachable. This concludes the proof that no marking in the set S defined above is reachable, hence the absence of this form of deadlocks is verified.*

Instead of checking for the absence of paths or cycles where every edge is traversed exactly once, it is often useful to be able to verify the absence of paths where very edge is traversed at most once. The corresponding set S' of markings can be easily obtained from S as follows: replace every set of the form $\{m_0 \oplus k_1 \cdot m_1 \oplus \cdots \oplus k_n \cdot m_n \mid k_i \in \mathbb{N}_0\}$ by $\{m \mid m \geq m_0\}$, i.e., only the minimal elements are kept. To check whether a marking of S' is reachable is a simpler problem called the coverability problem for Petri nets and can be checked via so-called coverability graphs, although for efficiency reasons it can also be useful to employ the approximative methods described above.

6 Conclusion.

We have shown how to use the concept of semilinear sets to verify the absence of certain Euler paths in all reachable graphs of a GTS. In the future we plan to generalize this technique by using context-free grammars instead of regular expressions. This should also lead to a broader applicability of the presented method. We believe that an important application area of this technique is to check for the absence of deadlocks, which involves structural aspects as well as quantitative aspects. The structural information can be obtained from the graph underlying the Petri graph, whereas the quantitative information (i.e., counting the number of edges) can be derived from the Petri net component.

On the practical side, checking whether a reachable marking is contained in a semilinear set can be done by solving linear inequations over the natural numbers in order to find suitable traps. Apart from traps other invariants (such as the marking equation) can be employed. For this task efficient tools such as `lp_solve` are available (see also [20] which describes an implementation).

Acknowledgements: I would like to thank Volker Diekert, Javier Esparza, Paolo Baldan and Arwed von Merkatz for interesting discussions on the topics of this paper. Furthermore I would like to thank Vitali Kozioura for developing and maintaining the AUGUR tool.

Bibliography

[1] Parosh Aziz Abdulla, Bengt Jonsson, Mats Kindahl, and Doron Peled. A general approach to partial order reductions in symbolic verification. In *Proc. of CAV '98*, pages 379–390. Springer, 1998. LNCS 1427.

[2] Paolo Baldan, Andrea Corradini, and Barbara König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR '01*, pages 381–395. Springer-Verlag, 2001. LNCS 2154.

[3] Paolo Baldan, Andrea Corradini, Barbara König, and Alberto Lluch Lafuente. A temporal graph logic for verification of graph transformation systems. In *Proc. of WADT '06 (Workshop on Algebraic Development Techniques)*, pages 1–20. Springer, 2007. LNCS 4409.

[4] Paolo Baldan and Barbara König. Approximating the behaviour of graph transformation systems. In *Proc. of ICGT '02 (International Conference on Graph Transformation)*, pages 14–29. Springer-Verlag, 2002. LNCS 2505.

[5] Paolo Baldan, Barbara König, and Bernhard König. A logic for analyzing abstractions of graph transformation systems. In *Proc. of SAS '03 (International Static Analysis Symposium)*, pages 255–272. Springer-Verlag, 2003. LNCS 2694.

[6] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation—part I: Basic concepts and double pushout approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, chapter 3. World Scientific, 1997.

[7] Fernando Luís Dotti, Luciana Foss, Leila Ribeiro, and Osmar Marchi Santos. Verification of distributed object-based systems. In *Proc. of FMOODS '03*, pages 261–275. Springer, 2003. LNCS 2884.

[8] Javier Esparza and Stephan Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16:159–189, 2000.

[9] Alain Finkel and Phillipe Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[10] Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge, and check - new algorithms for the coverability problem of WSTS. In *Proc. of FSTTCS '04*, pages 287–298. Springer, 2004. LNCS 3328.

[11] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[12] Barbara König and Vitali Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. of TACAS '06*, pages 197–211. Springer, 2006. LNCS 3920.

[13] Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[14] Wolfgang Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.

[15] Arend Rensink and Dino Distefano. Abstract graph transformation. In *Proc. of SVV '05 (3rd International Workshop on Software Verification and Validation)*, volume 157.1 of *ENTCS*, pages 39–59, 2005.

[16] Christophe Reutenauer. *Aspects mathématiques des réseaux de Pétri*. Masson, 1989.

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

[18] Claus Schröter. *Halbordnungs- und Reduktionstechniken für die automatische Verifikation von verteilten Systemen*. PhD thesis, Universität Stuttgart, 2005.

[19] Dániel Varró. Towards symbolic analysis of visual modeling languages. In *Workshop on Graph Transformation and Visual Modeling Techniques '02*, volume 72 of *ENTCS*. Elsevier, 2002.

[20] Arwed von Merkatz. Analyse von Graphtransformationssystemen mit Hilfe von Petrinetzen und Logiken. Master's thesis, Universität Stuttgart, July 2006. No. 2442.