Verification of Graph Transformation Systems with Context-Free Specifications*

Barbara König[1] and Javier Esparza[2]

[1] Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany
[2] Fakultät für Informatik, Technische Universität München, Germany

Abstract. We introduce an analysis method for graph transformation systems which checks that certain forbidden graphs are not reachable from the start graph. These forbidden graphs are specified by a context-free graph grammar. The technique is based on the approximation of graph transformation systems by Petri nets and on semilinear sets of markings. Especially we exploit Parikh's theorem which says that the Parikh image of a context-free grammar is semilinear. An important application is deadlock analysis for interaction nets and we specifically show how to apply the technique to an infinite-state dining philosopher's system.

1 Introduction

In recent years there have been several approaches directed specifically at the verification of graph transformation systems, for instance [21,10,24,1,2,4,22]. They usually address the following problem: Given a graph transformation system $\mathcal{T}$ generating a set $\mathit{reach}(\mathcal{T})$ of reachable graphs, and a set $\mathcal{F}$ of forbidden graphs, is the intersection $\mathit{reach}(\mathcal{T}) \cap \mathcal{F}$ empty? This problem is undecidable even for the case in which the set $\mathcal{F}$ contains one single graph, and so all approaches approximate the set $\mathit{reach}(\mathcal{T})$ in some way. In this paper we further develop the approach of [1,2], which—given $\mathcal{T}$—constructs a Petri net whose reachable markings encode a set $\mathcal{L}$ of graphs satisfying $\mathcal{L} \supseteq \mathit{reach}(\mathcal{T})$. So emptiness of $\mathcal{L} \cap \mathcal{F}$ implies emptiness of $\mathit{reach}(\mathcal{T}) \cap \mathcal{F}$.

The decidability, complexity, and practicality of checking emptiness of $\mathcal{L} \cap \mathcal{F}$ depend on the characteristics of the set $\mathcal{F}$. If $\mathcal{F}$ is defined as the set of all graphs containing a given subgraph, then emptiness reduces to the coverability problem of Petri nets. In [3] we studied the case in which $\mathcal{F}$ is the set of models of a formula in first-order or second-order monadic logic on graphs.

Unfortunately, many forbidden sets of interest for applications are not expressible in first-order or monadic second-order logic. This follows from the fact that all expressible sets are recognizable in the sense of Courcelle [7], while the forbidden sets are often not. For instance counting constraints usually specify non-recognizable languages. These sets can, however, often be described by (a

* Research supported by DFG project SANDS.

H. Ehrig et al. (Eds.): ICGT 2010, LNCS 6372, pp. 107–122, 2010.

slight variant of) context-free graph grammars. In this paper we propose an extended emptiness check for this class.

The main ideas behind the new check can already be introduced for transformation systems acting on words. Assume that $\mathcal{T}$ is such a system. The approximation of the word language $\mathcal{T}$ can be intuitively seen as equivalent to a finite automaton A and a *counting constraint* C.[1] The approximation $\mathcal{L}$ is given by $\mathcal{L} = L(A) \cap L(C)$, i.e., the words of $\mathcal{L}$ are those accepted by A and satisfying C. Later such counting constraints will be specified via the reachable markings of a Petri net.

Assume further that $\mathcal{F}$ is the language of forbidden words generated by a context-free word grammar G, i.e., $\mathcal{F} = L(G)$. Our task is to check the emptiness of $\mathcal{F} \cap \mathcal{L} = L(G) \cap L(A) \cap L(C)$. For this, we proceed in three steps:

(1) Compute a grammar G' accepting $L(G) \cap L(A)$ (using the well-known fact that the intersection of a context-free and a regular language is context-free). If $L(G') = \emptyset$, then terminate and conclude $\mathcal{F} \cap \mathcal{L} = \emptyset$, otherwise proceed with (2).
(2) Compute the strongest counting constraint C' satisfied by $L(G')$, specifically $L(G') \subseteq L(C')$. This is possible by Parikh's theorem, which states that the Parikh image of a context-free grammar is semilinear set, which can be formulated as a counting constraint.
(3) Check the emptiness of $L(C) \cap L(C')$ by deciding whether the conjunction $C \wedge C'$ is unsatisfiable. If it is unsatisfiable we have $\mathcal{F} \cap \mathcal{L} = \emptyset$, otherwise the intersection is non-empty.

The first contribution of the paper is an extension of this procedure to graph transformation systems. $\mathcal{T}$ is now a graph transformation system, and $\mathcal{F}$ is the set of graphs generated by a context-free graph grammar. The decidability of the satisfiability of $C \wedge C'$ is proved using results from Petri net theory.

While the decidability has some theoretical interest, the complexity of the resulting procedure is too high for practical purposes. Using techniques developed in [11], it is possible to over-approximate C' by a system of linear constraints C''. The satisfiability of $C \wedge C''$ can then be checked with appropriate solvers for linear programming (such as `lp_solve`).

An important application is to verify the absence of deadlocks, which often manifest themselves as "vicious cycles" (cycles of processes waiting for an action from another process). We show how to detect such cycles in the general setting of interaction nets and treat—as a concrete running example—an infinite-state version of the dining philosophers.

A simplified version of our results appeared as a workshop paper in [14]. That paper uses regular expressions instead of the more powerful context-free graph grammars employed here. Furthermore, in the current paper the theory is for the first time applied to the general setting of interaction nets.

[1] A constraint on the number of times that each letter can appear in a word of L; for instance, $(\#a - 2 \cdot \#b) \geq 3$ is a counting constraint for the alphabet $\{a, b\}$.

2 Preliminaries

2.1 Monoids and Semilinear Sets

Let M be a *monoid* with an associative binary operation written $m_1 m_2$ for $m_1, m_2 \in M$. Furthermore there exists a unit $1 \in M$. In this paper we will mainly consider the *free monoid* over a finite set S, denoted by S^*, and the *free commutative monoid* over S, denoted by $S^\oplus$. The former monoid consists of all *words* over S whereas the latter is isomorphic to the set of all mappings from S to $\mathbb{N}$, denoted by $\mathbb{N}^S$. Its elements will also be called *multisets*.

The *rational sets* of a monoid M are inductively defined as follows:

- Every finite subset of M is rational.
- Whenever A and B are rational sets, then $A \cup B$ and $AB = \{ab \mid a \in A, b \in B\}$ are rational sets.
- Whenever A is a rational set, then $A^* = \{a_1 \ldots a_n \mid n \geq 0,\, a_1, \ldots, a_n \in A\}$ is a rational set. Alternatively one can define A^* as the intersection of all sub-monoids of M containing A.

A *semilinear set* of a monoid M is of the form $\bigcup_{i \in I} \{a_i\} B_i^*$, where $a_i \in M$ and the B_i are finite subsets of M and I is a finite index set. Sets of the form $\{a_i\} B_i^*$ are called *linear*. For commutative monoids, every rational set is also semilinear. Furthermore semilinear sets are closed under intersection and complement [12].

In the following we will denote the monoid operation and the Kleene closure by $\oplus$ whenever we are working in a commutative monoid. Furthermore for a function $f \colon S \to T$ we will denote by $f^* \colon S^* \to T^*$ and $f^\oplus \colon S^\oplus \to T^\oplus$ its obvious extensions to words and multisets.

Whenever $\mu \colon S^* \to S^\oplus$ is the canonical monoid morphism and $A \subseteq S^*$, then $\mu(A)$ is called the *Parikh* image of A in $S^\oplus$. The Parikh image of a rational set is always semilinear.

2.2 Petri Nets

A *Petri net* is given by a tuple $N = (S_N, T_N, {}^\bullet(), ()^\bullet, p_N, m_0)$ where S_N is a set of places, T_N is a set of transitions, ${}^\bullet(), ()^\bullet \colon T_N \to S_N^\oplus$ assign to each transition its pre-set and post-set (where both are multisets), $p_N \colon T_N \to \Lambda$ assigns an action label to each transition and $m_0 \in S_N^\oplus$ is the initial marking.

Reachability and coverability in Petri nets is defined as usual. The set of all reachable markings of a net N is denoted by *reach*(N).

Proposition 1. *Let N be a Petri net where S_N is the set of places. Given a semilinear set A of $S_N^\oplus$ it is decidable whether there exists a reachable marking m of N that is contained in A.*

The proof is based on a reduction to the reachability problem for Petri nets. But although this problem is decidable it is not known to be primitive recursive and quite impossible to solve in practice. Hence, we aim at solving the reachability problem in an approximative way by solving systems of linear constraints, as detailed at the end of Section 3.

2.3 Hypergraphs

We assume that Λ is a fixed and finite set of labels and that each label $\ell \in \Lambda$ is equipped with an arity $ar(\ell) \in \mathbb{N}$.

Definition 1 (Hypergraph, hypergraph morphism). *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.*

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* $\varphi_V^*(c_G(e)) = c_{G'}(\varphi_E(e))$. *The morphism* φ *is called* edge-bijective *if* φ_E *is bijective.*

We will now introduce the language of all graphs that can be mapped homomorphically to a given graph. This is related to regular (word) languages, which intuitively contain all graphs consisting of a single path that can be mapped homomorphically to a finite automaton (taking into account also initial and final states). However note that the recognizable graph languages of Courcelle are more general.

Definition 2 (Language induced by a graph). *Let* G *be a hypergraph. Then we denote by* $\mathcal{I}_G$ *the* language induced by G, *defined by*

$$\mathcal{I}_G = \{G' \mid \exists\, \varphi\colon G' \to G\}.$$

For later use we need the following notion of node fusion.

Definition 3 (Closure under node fusion). *Let* $\mathcal{L}$ *be a graph language. By* $nf(\mathcal{L})$ *we denote the language obtained from* $\mathcal{L}$ *by fusing arbitrary nodes, i.e.,*

$$\begin{aligned} nf(\mathcal{L}) &= \{G/\!\equiv \mid G \in \mathcal{L} \textit{ and} \equiv \textit{is an equivalence on } V_G\} \\ &= \{G' \mid \varphi\colon G \to G', G \in \mathcal{L}, \varphi \textit{ edge-bijective}\}. \end{aligned}$$

2.4 Graph Transformation Systems

We now define the notion of a graph transformation system. In order to simplify the analysis we focus on restricted rewriting rules which can not delete nodes (but nodes may eventually become isolated).

Definition 4 (Graph transformation system). *A* graph transformation system (GTS) $\mathcal{T} = (G_0, \mathcal{R})$ *consists of an initial graph* G_0 *and a set* $\mathcal{R}$ *of rewriting rules of the form* $r = (L, R, \alpha)$, *where* L, R *are graphs, called* left-hand side *and* right-hand side, *respectively, and* $\alpha\colon V_L \to V_R$ *is a function.*

A match *of a rewriting rule* r *in a graph* G *is a morphism* $\varphi\colon L \to G$ *which is injective on edges. We can apply* r *to a match in* G *obtaining a new graph* H, *written* $G \overset{r}{\Rightarrow} H$, *where the target graph* H *is defined as follows. Let* $\equiv$ *be the smallest equivalence on* V_R *satisfying* $\alpha(w_1) \equiv \alpha(w_2)$ *whenever* $\varphi(w_1) = \varphi(w_2)$

for two nodes $w_1, w_2 \in V_L$*. Furthermore let* $\overline{\varphi}\colon V_G \uplus V_R \to V_H$ *be defined as follows:*

$$\overline{\varphi}(v) = \begin{cases} v & \text{if } v \in V_G - \varphi(V_L) \\ [\alpha(w)]_\equiv & \text{if } v \in \varphi(V_L),\ v = \varphi(w) \\ [v]_\equiv & \text{if } v \in V_R \end{cases}$$

Then we can define

$$\begin{array}{rcl} V_H = (V_G - \varphi(V_L)) \uplus (V_R/\equiv) & & E_H = (E_G - \varphi(E_L)) \uplus E_R \\ e \in E_G - \varphi(E_L) & \Rightarrow & c_H(e) = \overline{\varphi}(c_G(e)) \quad l_H(e) = l_G(e) \\ e \in E_R & \Rightarrow & c_H(e) = \overline{\varphi}(c_R(e)), \quad l_H(e) = l_R(e) \end{array}$$

The set of all reachable graphs, i.e., graphs that can be derived from G_0 *via repeated rule application, will be denoted by* $reach(\mathcal{T})$*.*

The application of r to G at the match φ first removes from G the image of the nodes and edges of L. Then the graph G is extended by adding the new nodes and edges in R, which are connected accordingly. Observe that the nodes of R have to be grouped into equivalence classes, i.e. merged, depending on whether they are assigned to the same nodes in G. The notion of graph rewriting that we use is a special case of the double-pushout approach.

Example 1. As a running example we consider an infinite-state dining philosophers system. The start graph of the system is depicted in Figure 1 where we see three philosophers (indicated by label H, since they are hungry) and three forks (indicated by labels F or F'). Some edges will be pointing clockwise along the cycle, others (indicated by a prime) counter-clockwise. Note that for binary edges, i.e., for edges connected to exactly two nodes, the first node is indicated by an arrowhead.

In order to avoid deadlocks we use a technique similar to the one presented in [6] where a dining philosophers system is seen as a directed acyclic graph with philosophers as nodes and forks as directed edges. In our setting only a philosopher which is maximal wrt. the partial order established by the forks, i.e., a philosopher which has only forks pointing towards it, may start eating.

Figure 2 depicts the rules governing the behaviour of the dining philosophers. A hungry philosopher may take up a fork pointing towards it (rule *(Wait)*) and

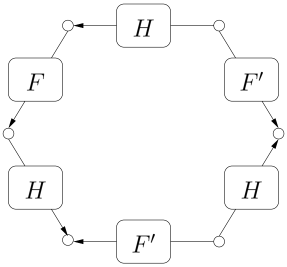


Fig. 1. Start graph of the infinite-state dining philosophers system

become a waiting philosopher (labelled W'). Subsequently a waiting philosopher may pick up another fork also pointing towards it and become an eating philosopher (rule *(Eat)*). The reason for modelling an eating philosopher by two edges is that we will later view this GTS as an instance of an interaction net system, where a left-hand side must always consist of exactly two edges. Since eating philosophers may react of their own impulse it is here necessary to represent them by two edges forming a valid redex. An eating philosopher may then either give back its forks in such a way that they are pointing away from it (rule *(Hungry)*) or replicate itself in order to create another hungry philosopher with another fork (rule *(Rep)*). This last rule makes the system infinite-state and hence non-trivial to analyze. Our aim is to show that no deadlock – which in this case manifests itself as a vicious cycle – will ever occur.

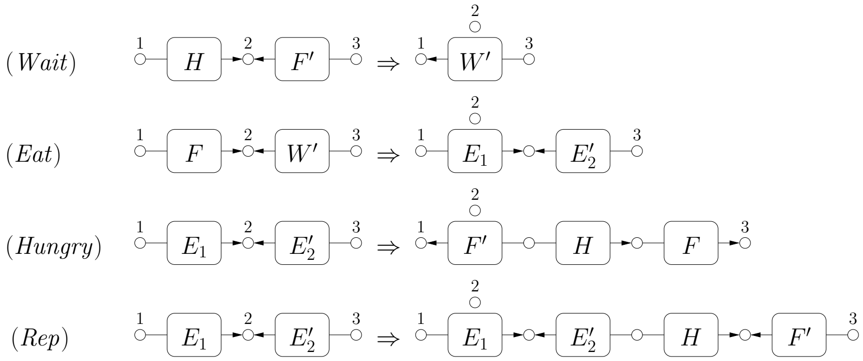


Fig. 2. Productions of the infinite-state dining philosophers system

2.5 Context-Free Graph Grammars

In order to characterize graph languages more complex than the ones of Definition 2, we introduce context-free graph grammars, also called hyperedge replacement grammars [13]. Similar to the case of word languages, the left-hand side of a context-free rule consists of a single hyperedge, whereas right-hand sides can be of arbitrary shape.

Definition 5 (Single hyperedge). *Let $\ell \in \Lambda$ be a label of arity $ar(\ell)$. Then $edge(\ell)$ is a hypergraph H which consists of a single edge e labelled ℓ and $ar(\ell)$ nodes such that the nodes in $c_H(e)$ are all pairwise different.*

Definition 6 (Context-free graph grammar). *Assume that the set of labels Λ is the disjoint union of Λ_T (the set of* terminal labels*) and Λ_N (the set of* non-terminal labels*). A* context-free graph grammar *$\mathcal{G}$ is a special graph transformation system where*

- *for every rule $r = (L, R, \alpha)$ it holds that $L = edge(A)$ for some $A \in \Lambda_N$ and α is injective.*
- *the start graph G_0 is of the form $edge(S)$ for some $S \in \Lambda_N$ with $ar(S) = 0$.*

Definition 7 (Context-free graph language). *Let $\mathcal{G}$ be a context-free graph grammar. Then the* language *generated by $\mathcal{G}$, denoted by $\mathcal{L}(\mathcal{G})$, is defined as*

$$\mathcal{L}(\mathcal{G}) = \{G \in \mathit{reach}(\mathcal{G}) \mid \text{all labels of } G \text{ are contained in } \Lambda_T\}.$$

In analogy to word languages, $\mathcal{L}(\mathcal{G})$ is also called a context-free (graph) language.

Note that not every language induced by a graph according to Definition 2 is context-free. Especially the language of all hypergraphs is induced by the final hypergraph (i.e., a graph consisting of one node and one edge for every label), but it is not context-free (cf. [13]).

Furthermore, given a graph language $\mathcal{L}$ over a label set Λ_T, its *Parikh image* is $\{l_G^{\oplus}(E_G) \mid G \in \mathcal{L}\} \subseteq \Lambda_T^{\oplus}$, i.e., for every graph in the language take the multiset of its edge labels.

2.6 Approximating Graph Transformation Systems by Petri Nets

In this section we sketch the algorithm, introduced in [1], for the construction of a finite approximation of the unfolding of a graph transformation system. Of course there is a straightforward counting abstraction via a Petri net which just counts the number of occurrences of each edge, regardless of structure. However, this will be too coarse for our purposes and we use Petri nets over a hypergraph structure, so-called *Petri graphs*.

Definition 8 (Petri graphs). *Let $\mathcal{T} = (G_0, \mathcal{R})$ be a GTS. A* Petri graph P (over $\mathcal{T}$) *is a tuple (G, N) where*

- *G is a graph;*
- *$N = (E_G, T_N, {}^\bullet(), ()^\bullet, p_N, m_0)$ is an $\mathcal{R}$-labelled Petri net, where the set of places is E_G, i.e., the edge set of G;*
- *$m_0 = \iota^{\oplus}(E_{G_0})$ for a graph morphism $\iota : G_0 \to G$ (i.e., m_0 must properly correspond to the initial state of the GTS $\mathcal{T}$).*

A marking $m \in E_G^{\oplus}$ will be called reachable (coverable) *in P if it is reachable (coverable) from the initial marking in the Petri net underlying P.*

A marking m of a Petri graph can be seen as an abstract representation of a graph in the following sense.

Definition 9. *Let (G, N) be a Petri graph and let $m \in E_G^{\oplus}$ be a marking of N. The graph H* generated *by m, denoted by $\mathit{graph}(m)$, is defined as follows: $V_H = \{v \in V_G \mid \exists e \in m\, \exists i\colon (v = [c_G(e)]_i)\}$, $E_H = \{(e,i) \mid e \in m \wedge 1 \leq i \leq m(e)\}$, $c_H((e,i)) = c_G(e)$ and $l_H((e,i)) = l_G(e)$.*[2]

That is, we take only the nodes adjacent to a marked edge and make parallel copies of all edges according to their multiplicity (see also Example 2 below).

[2] Note that $[\tilde{s}]_i$ denotes the i-th element of the string $\tilde{s}$ and that $m(e)$ denotes the multiplicity of e in the multiset m.

Given a GTS $\mathcal{T} = (G_0, \mathcal{R})$, with some additional minor constraints on the format of rewriting rules (see [1,2]), we can construct a Petri graph approximation of $\mathcal{T}$, called *covering* and denoted by $\mathcal{U}(\mathcal{T})$. The (0-)covering is produced by the following (terminating) algorithm which generates a sequence $P_i = (G_i, N_i)$ of Petri graphs. (There is also an extension which produces k-coverings, having a better precision.)

1. $P_0 = (G_0, N_0)$, 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 P_i into P_{i+1}, giving precedence to folding steps.

   Unfolding. Find a rule $r = (L, R, \alpha) \in \mathcal{R}$ and a match $\varphi\colon L \to G_i$ such that $\varphi^{\oplus}(E_L)$ is coverable in P_i. Then extend P_i by "attaching" R to G_i according to α and add a transition t, labelled by r, describing the application of rule r.

   Folding. Find a rule $r = (L, R, \alpha) \in \mathcal{R}$ and two matches $\varphi, \varphi'\colon L \to G_i$ 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 $\equiv$.

The first construction above is an unfolding construction that "unwinds" a system, while still preserving its concurrent behaviour. Hence unfoldings can lead to very compact descriptions of the state space of a concurrent system. Here, we add so called folding steps to the unfolding construction in order to obtain a finite over-approximation, even if the GTS has an infinite state space. We lose information by merging nodes that would actually be distinct. The covering $\mathcal{U}(\mathcal{T})$ is an abstraction of the original GTS $(G_0, \mathcal{R})$ in the following sense.

Proposition 2 (Abstraction [3]). *Let $\mathcal{T} = (G_0, \mathcal{R})$ be a graph transformation system and let $\mathcal{U}(\mathcal{T}) = (G, N)$ be its covering where N has initial marking m_0. Then there exists a simulation $\mathcal{S} \subseteq \mathit{reach}(\mathcal{T}) \times \mathit{reach}(N)$ with the following properties:*

- *$(G_0, m_0) \in \mathcal{S}$;*
- *whenever $(G, m) \in \mathcal{S}$ and $G \overset{r}{\Rightarrow} G'$, then there exists a marking m', obtained from m by firing a transition labelled r, and $(G', m') \in \mathcal{S}$;*
- *for every $(G, m) \in \mathcal{S}$ there is an edge-bijective morphism $\varphi\colon G \to \mathit{graph}(m)$.*

From this one can easily deduce that the set of all reachable graphs is over-approximated by the set of all graphs generated by reachable markings where edges can be "pulled apart" at the nodes in an arbitrary way, where by "pulling apart" we mean the reverse of node folding introduced earlier:

$$\mathit{nf}^{-1}(\{\mathit{graph}(m) \mid m \in \mathit{reach}(N)\}) \supseteq \mathit{reach}(\mathcal{T}) \qquad (1)$$

Example 2. With the approximated unfolding technique sketched above we obtain the Petri graph in Figure 3. It has been computed automatically using the tool AUGUR 2 [16]. Note that edges of the graph and places of the net coincide and that arcs between places and transitions are drawn with dashed lines. The marking shown is the initial marking m_0 and the corresponding graph $graph(m_0)$ has two nodes and six edges. Two parallel F'-labelled and three parallel H-labelled edges are going from right to left whereas one F-labelled edge is pointing from left to right. This graph is an over-approximation of G_0 in the sense described above.

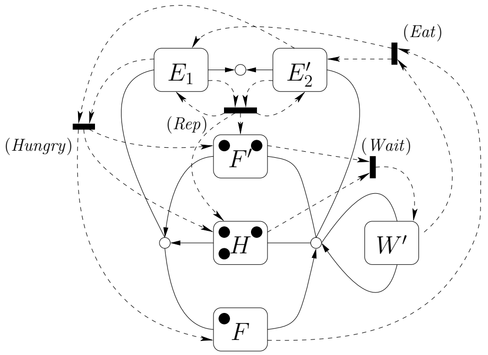


Fig. 3. A Petri graph approximating the dining philosophers system

3 Specifying Sets of Forbidden Graphs by Semilinear Sets

We assume that we are provided with a context-free graph grammar $\mathcal{G}$ that specifies the set of all "bad" or forbidden graphs, i.e., graphs representing error states. The set of forbidden graphs will not be the language generated by $\mathcal{G}$, but also all graphs G' that arise from graphs of $\mathcal{L}(\mathcal{G})$ by node fusion, i.e., we are interested in $nf(\mathcal{L}(\mathcal{G}))$. The introduction of node fusion is due to two reasons: first, the operator nf provides us with the possibility of specifying languages that can otherwise not be specified with a context-free grammar. For instance, the language of all graphs is not context-free (see [13]), but it can be obtained from a context-free language via node fusion. (Of course, this also means that other languages are excluded.) Second, our approximation method over-approximates with respect to the identity of nodes (see equation (1)), so if we find out—via the approximation—that a graph G is not reachable, the same is true for every preimage of G wrt. node fusion.

Given a graph transformation system $\mathcal{T}$, an over-approximating Petri graph $P = (G, N)$, i.e., a Petri graph satisfying equation (1), and a context-free graph grammar $\mathcal{G}$, our task is to construct a set S of markings of N which satisfies: *For a marking m of N it holds that $m \in S$ if and only if $graph(m) \in nf(\mathcal{L}(\mathcal{G}))$.*

If we can then show that no marking $m \in S$ is reachable in N, we know for sure that no graph $G \in nf(\mathcal{L}(\mathcal{G}))$ is reachable in $\mathcal{T}$. To prove this, assume that such a graph is reachable in $\mathcal{T}$. Then it holds that there exists a reachable marking m and an edge-bijective morphism $\varphi\colon G \to \mathit{graph}(m)$, i.e., $\mathit{graph}(m)$ is a "folded" variant of G. Furthermore $\mathit{graph}(m) \in \mathit{nf}(\mathcal{L}(\mathcal{G}))$ and hence as required above $m \in S$. But this is a contradiction, since we have checked that no marking of S is reachable in the net.

It will turn out that S is semilinear and it corresponds to the strongest counting constraint mentioned in the introduction. As a first step towards the construction of S we determine a context-free graph grammar that generates the intersection of a context-free graph language with a language induced by a graph, i.e., the set of graphs that can be mapped to a fixed graph G. Here G will usually be the graph component of a Petri graph.

As far as we know the underlying construction is original. It has similarities to the construction of the "cross-product" of a context-free grammar with a finite automaton. Furthermore there is a related result by Courcelle [7] who shows that the intersection of recognizable set and a equational set of graphs is always equational. Equational sets correspond to context-free graph languages.

Proposition 3. *For a context-free graph grammar $\mathcal{G}$ and a graph G there exists a context-free graph grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G}) \cap \mathcal{I}_G$.*

The construction used in the proof of the previous proposition is now the key to the construction of the required semilinar set S. Specifically, we can derive S from the grammar $\mathcal{G}'$ of Proposition 3, which implies that S is semilinear.

Proposition 4. *For a context-free graph grammar $\mathcal{G}$ and a graph G the following set $S_G^{\mathcal{G}}$ is a semilinear set of $E_G^{\oplus}$: $S_G^{\mathcal{G}} = \{m \in E_G^{\oplus} \mid \mathit{graph}(m) \in \mathit{nf}(\mathcal{L}(\mathcal{G}))\}$.*

Finally it is well-known that $S_G^{\mathcal{G}}$ is effectively computable and furthermore it follows from Proposition 1 that it is decidable whether there exists a reachable marking of a net that is contained in a given semilinear set. This would allow us to automatically verify whether $\mathit{graph}(m) \notin \mathit{nf}(\mathcal{L}(\mathcal{G}))$ for all reachable markings m, which—as detailed above—implies that no graph contained in $\mathit{nf}(\mathcal{L}(\mathcal{G}))$ is reachable in the graph transformation system $\mathcal{T}$. Figure 4 summarizes the technique.

However, checking whether no marking of a semilinear set is reachable is at least as costly as the reachability problem for Petri nets [19]. Since this problem is not known to be primitive recursive and all known algorithms are very complex,

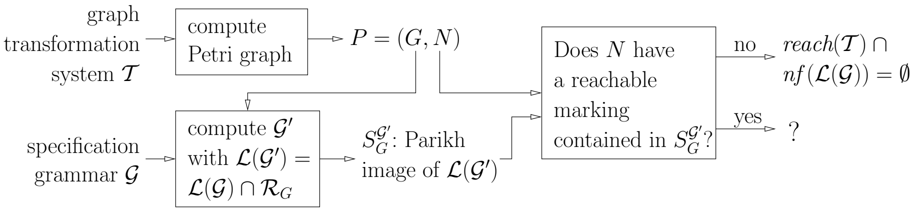

graph transformation system $\mathcal{T}$
compute Petri graph
$P = (G, N)$
Does N have a reachable marking contained in $S_G^{\mathcal{G}'}$?
no
$\mathit{reach}(\mathcal{T}) \cap \mathit{nf}(\mathcal{L}(\mathcal{G})) = \emptyset$
yes
?
specification grammar $\mathcal{G}$
compute $\mathcal{G}'$ with $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G}) \cap \mathcal{R}_G$
$S_G^{\mathcal{G}'}$: Parikh image of $\mathcal{L}(\mathcal{G}')$

Fig. 4. Summary of the technique

it will usually be necessary to resort to approximative reachability methods based on the marking equation and traps [11]. These techniques are based on the efficient solving of linear constraints over natural numbers, which can be handled using a tool such as `lp_solve`.

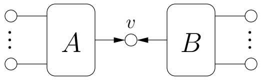
4 Characterizing Deadlocks of Interaction Nets

Interaction nets [17] are a special form of graph transformation systems with strong restrictions on the start graph and on the form of rules. In this setting one can ensure confluence and furthermore a deadlock (i.e., non-applicability of any rule) always manifests itself structurally in the presence of vicious cycles. We will here present a slight adaptation of interaction nets, especially we will differ by allowing isolated nodes, which however have no influence on further reductions.

In the following, the first node attached to a hyperedge will be called *principal node* and denoted by an arrow head. The *degree* of a node v is the number of "tentacles" attached to a node, i.e., $\deg(v) = |\{(e,i) \mid [c_G(e)]_i = v\}|$.

Definition 10 (Interaction net (system)). *A graph transformation system $\mathcal{T}$ is called an interaction net system whenever the following conditions hold:*

- *Every node of the start graph is either isolated or has degree* 2*. (Any such graph is called* interaction net*.)*
- *For a rewriting rule (L, R, α) the left-hand side L has the following form:*



We require that $\deg(\alpha(v)) = 0$ *and that v is the only node that α maps to $\alpha(v)$. For every other node v' in the right-hand side R it holds that:* $\deg(v') = 2 - |\alpha^{-1}(v')|$*. (That is, nodes which are not in the image of α have degree* 2 *and whenever two nodes of the left-hand side are merged, the resulting node has degree* 0*. The remaining nodes have degree* 1*.)*
Finally, for every pair $A, B \in \Lambda$ of labels there exists a rule with a left-hand side as depicted above.

Note that the second condition above is required in order to make sure that interaction nets are closed under rewriting.

All nodes in the left-hand side which are different from v and their images in the right-hand side are called *external.*

Example 3. The example graph transformation system introduced in Section 2.4 satisfies the conditions imposed by Definition 10, apart from the last. This last condition can be satisfied by adding dummy rules for the remaining left-hand sides. These dummy rules have no effect, i.e., a left-hand side with two edges (e.g., F, F') is replaced by the same left-hand side plus an isolated node.

It can then be shown that in an interaction net system every reachable graph has only nodes which are either isolated or have degree 2. Furthermore it can be shown (see [17]) that a graph that does not allow the application of a rule has a vicious cycle. In order to formally characterize vicious cycles we first need the following definition.

Definition 11 (C-path). *Let C be a set of triples (i, A, j), where $A \in \Lambda$ and $i, j \in \{1, \dots, ar(A)\}$, $i \neq j$. A C-*path *is a sequence $v_0, e_1, v_1, \dots, e_n, v_n$ of nodes and edges such that two nodes v_k, v_{k+1} are connected by the edge e_k where $[c_G(e_k)]_i = v_{k-1}$, $[c_G(e_k)]_j = v_k$ and $(i, l_G(e_i), j) \in C$. We also require that all edges e_k are distinct. A C-path is a C-*cycle *if additionally $v_n = v_0$.*

We call a C-path a principal *C-path whenever for all $(i, A, j) \in C$ it holds that either i=1 or j=1. It is a* clockwise *C-path if always j=1 and a* counterclockwise *C-path if always i=1. (Similar for C-cycles.)*

A clockwise C-cycle where $C = \{(i, A, 1) \mid A \in \Lambda, i \in \{2, \dots, ar(A)\}\}$ (i.e., C contains all possible triples) is also called vicious cycle*.*

Proposition 5 (Vicious cycles [17]). *An interaction net in which no reduction is possible is either the empty graph or it contains at least one vicious cycle.*

Note that this proposition depends heavily on the satisfaction of the last item of Definition 10 and hence, in our setting, on the existence of dummy rules.

Graphs not containing vicious cycles can be specified using hyperedge replacement grammars (and node fusion). However, if we do not have additional information about the potential form of the vicious cycles, a better and more local condition can be used in order to check for the absence of deadlocks. It is enough to check that for each reachable marking m there exists a node v such that the number of tokens placed on edges having v as principal node is strictly larger than the number of tokens placed on edges having v as non-principal node. The node v represents several nodes and the condition implies that at least one of them is the principal node of two edges. The negation of this requirement can be expressed as a semilinear set of markings, or, even simpler, directly encoded into a linear constraint and passed to a linear constraint solver.

Example 4. Note that the technique described above does not work for the running example with the Petri graph depicted in Figure 3. Specifically, it discovers a possible vicious cycle with labels W', W', H, F that can be reached after firing the transition representing rule (*Wait*) twice. However, this cycle is not among the reachable graphs and a modification of the technique is necessary.

5 A Refined Deadlock Analysis

If we know that all reachable interaction nets satisfy a certain invariant, we can omit deadlock configurations which are known to be unreachable and only specify the remaining situations via a context-free graph grammar. An invariant that works well in a number of examples is being $\{C_i\}_i$*-cycle-covered.*, i.e., a graph is covered by cycles $k_1, \dots, k_n$, where k_i is a C_i-cycle (for a set C_i, see

Definition 11), and all other cycles are trivial, which means that they consist of only one node.

It is possible to define sufficient conditions implying this invariant. Furthermore it follows from the invariant that an arising vicious cycle must be one of the C_i-cycles, and that the remaining edges are from the other cycles. This stronger non-local property can be described by a hyperedge replacement grammar.

A context-free grammar $\mathcal{G}_\ell$ specifying the forbidden graphs with a vicious C_ℓ-cycle either generates the empty graph or a clockwise C_ℓ-cycle or a counterclockwise C_ℓ-cycle for some index ℓ. In addition it must generate all edges present in C_m for $m \neq \ell$. This can be done with the context-free graph grammar, having nonterminals S, CW, CC, given in Figure 5. Then it is necessary to check that there is an empty intersection of the reachable graphs with each language $\mathcal{L}(\mathcal{G}_\ell)$.

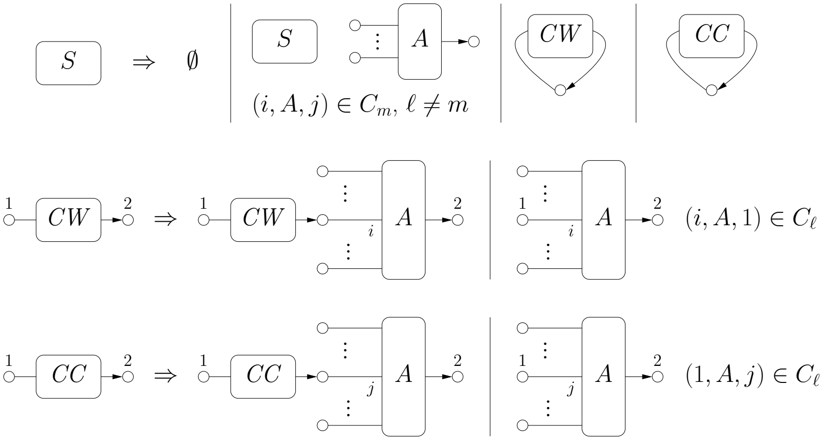


Fig. 5. A context-free graph grammar $\mathcal{G}_\ell$ for the generation of vicious C_ℓ-cycles

Example 5. In our running example we can easily check the following invariant, by inspection of the rules and the initial graph: every reachable graph is a C-cycle for

$$C = \{(1, H, 2), (1, F, 2), (1, E_1, 2), (2, F', 1), (2, W', 1), (2, E_2', 1)\}.$$

(So, in particular, every reachable graph is $\{C\}$-cycle-covered.) It follows that a vicious cycle of a reachable graph must necessarily be equal to the graph itself.

In this case, the context-free graph grammar generating all vicious C-cycles either generates a counterclockwise cycle consisting only of edge labels H, F, E_1 or a clockwise cycle consisting only of edge labels F', W', E_2'. Applying the procedure for computing semilinear sets of Section 3 to this grammar and the Petri graph of Figure 3 we obtain $S = \{H \oplus F\}^{\oplus} \cup \{W'\}^{\oplus}$ (where edges are denoted by their labels, since every label occurs at most once in our Petri graph). The first set corresponds to a situation where only H- and F-edges are present and their numbers are the same, i.e., to a vicious cycle of the form $H, F, H, F, \ldots$. The second set corresponds to the cycle $W', W', \ldots$.

No marking of S is reachable. This can be shown by setting up systems of linear constraints, one for each linear set. In our case it is sufficient to use the marking equation and the equations describing the semilinear set S above. No solution can be found, and so that this dining-philosophers system will never reach a deadlock state.

Finally, we can also specify as error graphs all graphs that allow only the application of the dummy rules introduced above, a more useful property to verify. Since C-cycles have been established as invariant, all reachable cycles can be represented as words over the alphabet C by cutting the cycle at an arbitrary point. The following regular expression r describes the language of all "good" cycles allowing the application of at least one non-trivial rule as shown in Figure 2:

$$\begin{aligned} r = {} & C^*(2,H,1)(1,F',2)C^* + C^*(2,F,1)(1,W',2)C^* + C^*(2,E_1,1)(1,E_2',2)C^* \\ & + (1,F',2)C^*(2,H,1) + (1,W',2)C^*(2,F,1) + (1,E_2',2)C^*(2,E_1,1) \end{aligned}$$

Using standard techniques one can compute a regular expression for the complement language (C-cycles allowing only the application of trivial rules) and turn it into a context-free graph grammar. Applying our techniques we obtain the same semilinear set S as above and the same reasoning applies.

6 Conclusion

We have shown how to analyze a graph transformation system where the set of forbidden graphs is specified by a context-free graph grammar. This is done by using Parikh's theorem which states that the Parikh image of a context-free language is semi-linear. Semilinear sets can then be expressed using linear constraints. Similarly, the specification of the error graphs can be written as linear constraints. If the resulting constraint system has no solution, we can infer that no error graph is reachable.

We use graph transformation to model and analyze systems with dynamically evolving communication structures. Several papers in this area concentrate on identifying decidable classes. Lammich et al. study dynamic pushdown networks, in which communication between parallel threads is either abstracted away or limited (see e.g. [18]), and Meyer describes a decidable fragment of the π-calculus in [20], subsuming decidability results of previous papers. None of these formalisms, however, can model the examples of this paper. Results on parametrized verification cannot be directly applied either, because in those approaches the number of processes stays fixed during the computation, and the communication structure is restricted to trees or other structures.

We follow the line of overapproximating the set of reachable states, which in our case are graphs. Apart from the work on the verification of graph transformation mentioned in the introduction, related work along this line is shape analysis [23] and, more recently, separation logic [9]. Both, however, are mainly specialized towards the verification of specific pointer structures, such as singly-linked or doubly-linked lists, rather than arbitrary graphs.

The idea of using Parikh's theorem for overapproximating context-free structures has been used by Deutsch [8] and by Bouajjani et al. [5] in settings different from ours.

The method described in the paper has been partially implemented in the tool Augur 2 [16]. Especially, we have implemented the computation of the over-approximating Petri graph and the computation of the semi-linear set from a context-free grammar. We plan to investigate techniques to derive linear constraints directly from the context-free grammar, without going via semi-linear sets, and techniques to automatically generate invariants like the ones obtained here through vicious cycles.

Finally, verification may also be unsuccessful because the Petri net over-approximates too coarsely. We have developed a counterexample-guided abstraction refinement technique [15] that could be adapted to the setting of this paper.

Acknowledgements. We would like to thank Volker Diekert, Nicolas Relange, Paolo Baldan, Arwed von Merkatz, Stefan Kiefer, Vitali Kozioura and Salil Joshi for helpful discussions on relevant topics, for earlier related work and for their help with the implementation.

References

1. Baldan, P., Corradini, A., König, B.: A static analysis technique for graph transformation systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 381–395. Springer, Heidelberg (2001)
2. Baldan, P., König, B.: Approximating the behaviour of graph transformation systems. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 14–29. Springer, Heidelberg (2002)
3. Baldan, P., König, B., König, B.: A logic for analyzing abstractions of graph transformation systems. In: Cousot, R. (ed.) SAS 2003. LNCS, vol. 2694, pp. 255–272. Springer, Heidelberg (2003)
4. Bauer, J., Wilhelm, R.: Static analysis of dynamic communication systems by partner abstraction. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 249–264. Springer, Heidelberg (2007)
5. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. In: Proc. of POPL 2003, pp. 62–73. ACM, New York (2003)
6. Chandy, K.M., Misra, J.: The drinking philosophers problem. ACM Transactions on Programming Languages and Systems 6(4), 632–646 (1984)
7. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, ch. 5. Foundations, vol. 1. World Scientific, Singapore (1997)
8. Deutsch, A.: Interprocedural may-alias analysis for pointers: Beyond k-limiting. In: Proc. of PLDI 1994. SIGPLAN Notices, vol. 29(6), pp. 230–241. ACM, New York (1994)
9. Distefano, D., O'Hearn, P.W., Yang, H.: A local shape analysis based on separation logic. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 287–302. Springer, Heidelberg (2006)

10. Dotti, F.L., Foss, L., Ribeiro, L., Marchi Santos, O.: Verification of distributed object-based systems. In: Najm, E., Nestmann, U., Stevens, P. (eds.) FMOODS 2003. LNCS, vol. 2884, pp. 261–275. Springer, Heidelberg (2003)
11. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design 16, 159–189 (2000)
12. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas, and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
13. Habel, A.: Hyperedge Replacement: Grammars and Languages. In: Habel, A. (ed.) Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992)
14. König, B.: Graph transformation systems, Petri nets and semilinear sets: Checking for the absence of forbidden paths in graphs. In: Proc. of PNGT 2006. Electronic Communications of the EASST, vol. 2 (2007)
15. König, B., Kozioura, V.: Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 197–211. Springer, Heidelberg (2006)
16. König, B., Kozioura, V.: Augur 2—a new version of a tool for the analysis of graph transformation systems. In: Proc. of GT-VMT 2006. ENTCS, vol. 211, pp. 201–210. Elsevier, Amsterdam (2006)
17. Lafont, Y.: Interaction nets. In: Proc. of POPL 1990, pp. 95–108. ACM Press, New York (1990)
18. Lammich, P., Müller-Olm, M., Wenner, A.: Predecessor sets of dynamic pushdown networks with tree-regular constraints. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. LNCS, vol. 5643, pp. 525–539. Springer, Heidelberg (2009)
19. Mayr, E.W.: An algorithm for the general Petri net reachability problem. SIAM J. Comput. 13(3), 441–460 (1984)
20. Meyer, R.: On boundedness in depth in the pi-calculus. In: Proc. of IFIP TCS 2008. IFIP, vol. 273, pp. 477–489. Springer, Heidelberg (2008)
21. Rensink, A., Distefano, D.: Abstract graph transformation. In: Proc. of SVV 2005. ENTCS, vol. 157.1, pp. 39–59 (2005)
22. Rieger, S., Noll, T.: Abstracting complex data structures by hyperedge replacement. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 69–83. Springer, Heidelberg (2008)
23. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. TOPLAS (ACM Transactions on Programming Languages and Systems) 24(3), 217–298 (2002)
24. Varró, D.: Towards symbolic analysis of visual modeling languages. In: Proc. of GT-VMT 2002. ENTCS, vol. 72. Elsevier, Amsterdam (2002)