On the Decidability Status of Reachability and Coverability in Graph Transformation Systems*

Nathalie Bertrand[1], Giorgio Delzanno[2], Barbara König[3], Arnaud Sangnier[4], and Jan Stückrath[3]

1 **Inria Rennes Bretagne Atlantique, France**
2 **Università di Genova, Italy**
3 **Universität Duisburg-Essen, Germany**
4 **LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, CNRS, France**

Abstract

We study decidability issues for reachability problems in graph transformation systems, a powerful infinite-state model. For a fixed initial configuration, we consider reachability of an entirely specified configuration and of a configuration that satisfies a given pattern (coverability). The former is a fundamental problem for any computational model, the latter is strictly related to verification of safety properties in which the pattern specifies an infinite set of bad configurations. In this paper we reformulate results obtained, e.g., for context-free graph grammars and concurrency models, such as Petri nets, in the more general setting of graph transformation systems and study new results for classes of models obtained by adding constraints on the form of reduction rules.

1 Introduction

Graph transformation systems (GTS) form an intuitive, but precise modelling framework, which has been extensively studied since its introduction in the 1970's. A graph transformation system consists of an initial graph and a set of reduction rules, which rewrite graphs and thus generate a transition system with graphs as states. Applications come from diverse areas such as the specification of UML model transformation [14] or encoding of process calculi [4].

In the past there has been a strong focus on questions of (categorical) semantics and expressiveness, while algorithmic issues received much less attention. Especially, different from related formalisms such as Petri nets [15], there is no systematic study of decidability results for graph transformation systems.

Taking inspirations from recent work on concurrency models [5, 8, 19, 20, 23], we consider here decidability issues for two fundamental problems: *reachability* and *coverability* of a given graph G_f from an *initial graph* G_0. The first problem requires the existence of a computation from G_0 to (a graph isomorphic to) G_f. The second one requires the existence of a computation from G_0 to some graph G that includes G_f as a subgraph.

While it is straightforward to determine the decidability status of these problems for general graph transformation systems, where the problems are both undecidable, and finite-state graph transformations, where they are both decidable, there are several other classes of infinite-state systems for which the question can be naturally asked. In this paper we systematically analyse reachability and coverability for several subclasses of GTS obtained as syntactic restrictions of reduction rules.

* Research partially supported by DFG project GaReV.

For special classes of GTS in which rules never change the underlying structure, we also consider an existential formulation of the coverability problem in which the initial configuration is an unknown variable to be discovered. This problem naturally models parameterized verification questions for concurrent systems with a static topology [9].

With our analysis, we combine within a unified framework both known results coming from fields such as context-free graph grammars, concurrency and verification, with new ones that we obtain by considering restrictions such as non-deletion of nodes, well-structuredness of the rewriting relation, and relabelling reductions only. An algorithmic view of graph transformation systems could open new research directions in the field of verification of infinite-state systems.

Due to length restrictions this paper contains mostly proof sketches. A version of this paper including all proofs in full length is published as a technical report [3].

2 What is a Graph Transformation System?

A graph transformation systems (GTS) is defined by a collection of reduction rules that can be used to dynamically modify the structure of an initial hypergraph. Reduction rules can conveniently be defined as graph morphisms. To formalize these ideas, we next define (labelled) hypergraphs – in the following simply called graphs – and graph morphisms.

▶ **Definition 1.** Let Λ be a finite set of edge labels and $ar\colon \Lambda \to \mathbb{N}_0$ a function that assigns an arity to each label. A *(Λ-)hypergraph* is a tuple (V_G, E_G, c_G, l_G^E) where V_G is a finite set of nodes, E_G is a finite set of edges, $c_G\colon E_G \to V_G^*$ is a connection function and $l_G^E\colon E_G \to \Lambda$ is an edge labelling function. We require that $|c_G(e)| = ar(l_G^E(e))$ for each edge $e \in E_G$.

An edge e is called *adjacent* to a node v if v occurs in $c_G(e)$.

We remark here that we consider graphs in which hyperedges have a fixed arity determined by their label. Directed labelled graphs are a special case of hypergraphs where every edge label has arity 2 and every sequence $c_G(e)$ is of length two.

▶ **Definition 2.** Let G, G' be (Λ-)hypergraphs. A *partial hypergraph morphism* (or simply *morphism*) $\varphi\colon G \rightharpoonup G'$ consists of a pair of partial functions $(\varphi_V : V_G \rightharpoonup V_{G'}, \varphi_E : E_G \rightharpoonup E_{G'})$ such that for every $e \in E_G$ it holds that $\varphi_V(c_G(e)) = c_{G'}(\varphi_E(e))$ whenever $\varphi_E(e)$ is defined. Furthermore if a morphism is defined on an edge, it must be defined on all nodes adjacent to it.

A morphism is called *label-preserving* if in addition $l_G^E(e) = l_{G'}^E(\varphi_E(e))$ for all $e \in E_G$, where $\varphi_E(e)$ is defined. *Total morphisms* are denoted by an arrow of the form $\to$.

In the following we drop the subscripts and write φ instead of φ_V and φ_E. Note that in the literature graph morphisms are usually label-preserving. Here we are more flexible in our definition, since we want to use graph morphisms also in order to define relabellings.

Isomorphism, Subgraphs, and Minors

Two graphs G, H are called *isomorphic* if there is a bijective, label-preserving morphism from G to H. In the rest of the paper we consider the following graph orderings over hypergraphs: $G \sqsubseteq_s H$, if G is a *subgraph* of H, namely, there exists a total, injective and label-preserving morphism from G to H, and $G \sqsubseteq_m H$ if G is a *minor* of H, i.e., G can be obtained from H by (iterative) node deletion, edge deletion and edge contraction. *Edge contraction* for hypergraphs means that the edge is deleted and its adjacent nodes are merged arbitrarily, i.e. merging nodes according to any partition on the adjacent nodes will result in a valid

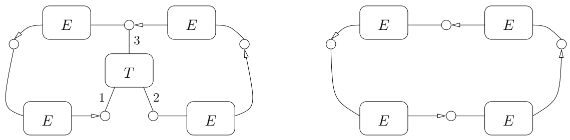


Figure 1 A graph (left) and its minor (right).

edge contraction (see also [22]). Fig. 1 shows an example of an edge contraction: the 3-ary hyperedge labelled T (where the nodes attached to the edge are numbered 1, 2, 3 in their respective order) is contracted with partition $\{1,2\}$, $\{3\}$, which means that we obtain a circle of E-edges.

Based on results in [22] it can be shown that the minor order defined above is a well-quasi order[1] on the set of all hypergraphs. The minor order on hypergraphs was first differently introduced in [17], contracting *all* nodes attached to a hyperedge. However, unfortunately it does not follow from the results in [22] that the order studied in [17] is a wqo. By using the minor order defined above instead, the results of [17] can be recovered, see [18] for a corrected version. Furthermore the subgraph order is a wqo on a restricted set of graphs (for more details see Section 3.5). Note that if G is a subgraph of H, it is also a minor, but not necessarily vice versa.

Graph Rewriting

We now define the rewriting mechanism, a slight extension of single-pushout rewriting (SPO) [13]. One of the most important features of SPO is the handling of so-called "dangling" edges: whenever a node is deleted, all edges attached to it have to be removed as well, also those which are not explicitly deleted. Different from standard SPO we allow as rules partial morphisms which are not necessarily label-preserving. In such a case the corresponding edge is preserved and relabelled. In the definition below this is simulated by removing the old edge and adding a new edge with the modified label. This will be especially important in Section 3.7 where we consider node and edge relabelling as well.

▶ **Definition 3.** A *rewriting rule* is a partial morphism $r\colon L \rightharpoonup R$, where L is called left-hand side and R right-hand side. A *match* (of r) is a total, injective and label-preserving morphism $m\colon L \to G$.[2]

Given a rule r and a match m, a *rewriting step* or an application of the rule to the graph G, results in a graph H (symbolically $G \Rightarrow_{\mathcal{R}} H$), which is defined as follows. Let E_D be the set of edges $e \in E_G$ which are adjacent to a node $m(v)$ where $r(v)$ is undefined (so-called *dangling edges*). We define the node and edge sets of H as follows: $V_H = V_G \backslash V_L \cup V_R$, $E_H = E_G \backslash (E_L \cup E_D) \cup E_R$.

Define a mapping $\tilde{r}\colon V_G \rightharpoonup V_H$ such that $\tilde{r}(v) = v$ if $v \in V_G \backslash V_L$, and $\tilde{r}(v) = r(v)$ if $v \in V_L$. Finally, define the attachment and edge labelling functions of H:

$$c_H(e) = \begin{cases} c_R(e) & \text{if } e \in E_R \\ \tilde{r}(c_G(e)) & \text{if } e \in E_G \backslash (E_L \cup E_D) \end{cases} \qquad l_H^E(e) = \begin{cases} l_R^E(e) & \text{if } e \in E_R \\ l_G^E(e) & \text{if } e \in E_G \backslash (E_L \cup E_D) \end{cases}$$

[1] A quasi order (on graphs) is a well-quasi order (wqo) if in every infinite sequence $G_1, G_2, \dots$ of graphs there are indices $i < j$ with $G_i \sqsubseteq G_j$.

[2] Since m is injective we can assume that it acts as the identity on nodes and edges on which it is defined, i.e., $m(v) = v$ and $m(e) = e$ for $v \in V_L, e \in E_L$. In addition we assume that the set of nodes and edges in R not in the image of r are disjoint from the set of nodes and edges of G.

Intuitively, we can think of this as follows: L is a subgraph of G, all items of L whose image is undefined under r are deleted, the new items of R are added, merged and connected as specified by r. Whenever a node is deleted, all adjacent edges will be deleted as well. In addition, edges are relabelled as specified by r.

Note that for Definition 3 it would be sufficient to define r solely on nodes and to create and recreate edges instead of preserving them. However, to be consistent with later sections, especially Section 3.7, we allow that r is also defined on edges.

▶ **Example 4.** In the following we introduce an (erroneous) termination detection protocol on a ring, modelled as a GTS to illustrate a rewriting step as well as the differences between the classes of GTS studied in later sections.

The protocol is initialised with a ring structure containing an active process, a passive process and a passive detector as shown in Figure 2. The first three rules presented in Figure 3 allow normal and detector processes to deactivate themselves (3a), activate other processes (3b) and generate new processes (3c). The last three rules shown in Figure 3 allow detector processes to generate termination messages (3d), normal processes to forward these messages (3e) and detector processes to generate a termination flag after receiving a termination message (3f).

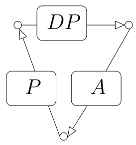
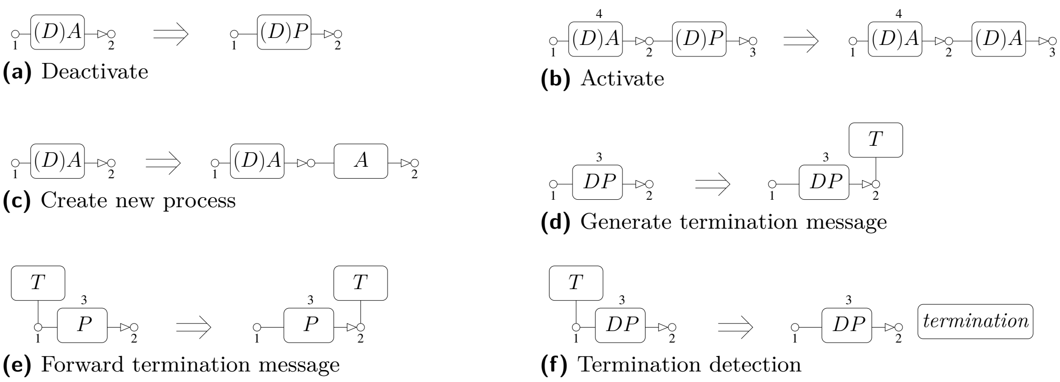


Figure 2 Initial graph

Edges are represented as boxes with rounded edges and binary edges are drawn as directed edges, indicating the order of the nodes wrt. the edge. The example includes 0-ary hyperedges (with label *termination*), unary hyperedges (with label T) and binary hyperedges with labels (A, P, DA, DP). The label $(D)A$ thereby represents both the label A and DA (label $(D)P$ is uses analogously), i.e. the rule in Figure 3a can either rewrite an A-edge to a P-edge or a DA-edge to a DP-edge. In a rule, an item (node or edge) in the left-hand side numbered i is mapped to the corresponding item on the right-hand side.

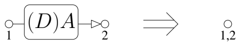
This protocol is supposed to detect whether all processes are passive and generate a termination flag in that case. However, by means of the methods of Section 3.5 it can be proven erroneous. Figure 5 represents the pattern which a correct protocol will never produce, that is an active process and a termination flag. Hence we can refute the correctness by showing that a graph containing the pattern is reachable.



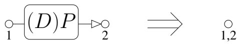
(a) Deactivate
(b) Activate
(c) Create new process
(d) Generate termination message
(e) Forward termination message
(f) Termination detection

Figure 3 The basic rules of a termination detection protocol

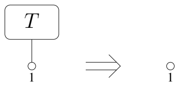
In order to model a lossy system, we add rules such that active (4a) and passive (4b) processes can leave the ring and the termination message (4c) and flag (4d) may be lost.



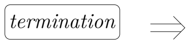
(a) Active process leaves



(b) Passive process leaves



(c) Message lost



(d) Termination flag lost

Figure 4 Additional rules simulating a lossy system

Figure 6 exemplarily shows a rewriting step where the deactivation rule is applied to the initial graph. The active process is thereby deleted and replaced by an identical but passive process, resulting in the bottom right graph, where all processes are passive.

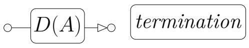


Figure 5 The error graph

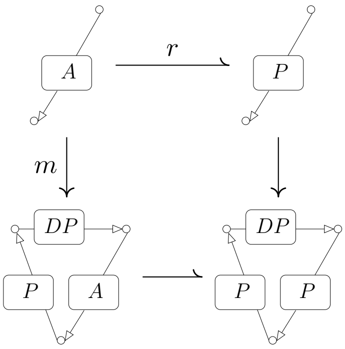


Figure 6 A rewriting example

3 An Algorithmic Study of Reachability and Coverability

As mentioned in the introduction, we focus our attention on fundamental decision problems for studying computational properties of a model, namely reachability and coverability. The following definitions are given modulo isomorphism.

▶ **Definition 5.** Given a finite set of rewriting rules $\mathcal{R}$, an initial graph G_0 and a final graph G_f, the Reachability Problem is defined as follows: does $G_0 \Rightarrow^*_{\mathcal{R}} G_f$ hold?

▶ **Definition 6.** Given a finite set of rewriting rules $\mathcal{R}$, an initial graph G_0 and a final graph G_f, the Coverability Problem is defined as follows: is there a graph H such that $G_0 \Rightarrow^*_{\mathcal{R}} H$ and $G_f \sqsubseteq_s H$?

We will now study decidability and undecidability results by extending existing results (e.g., for context-free grammars, GTS and Petri nets, and well-structured transition systems) with new results for several fragments of GTS. The conclusion (Section 5) gives an overview over the various cases in a single diagram. We first recall that the reachability and the coverability problem are both undecidable in the general case. This is a known result. It is quite straightforward to encode Turing machines into graph rewriting and either problem admits a reduction from the halting problem.

Another immediate results follows from restricting the set of reachable graphs to be finite. If it is known that only finitely many graphs up to isomorphism are reachable from

G_0, the reachability and coverability problems are decidable for the given GTS. Indeed we just have to enumerate all graphs up to isomorphism and check whether they are isomorphic or in relation to G_f. While the result follows quite trivially, an efficient implementation is more demanding (see for instance the GROOVE tool [21]).

3.1 Symbolic Backward Reachability Algorithm

Symbolic backward analysis turns out to be a convenient tool to answer coverability queries. We present here a generic variant that we apply in some of the proofs in the rest of the paper.

The key idea is to use a graph H as a symbolic representation of its upward closure wrt. some subsumption relation $\sqsubseteq$, namely the infinite set of graphs $up(H) = \{G \mid H \sqsubseteq G\}$. The subsumption relation is in our case either the subgraph inclusion ($G \sqsubseteq_s H$) or the graph minor ordering ($G \sqsubseteq_m H$). For an upward closed set X, we define $\lfloor X \rfloor$ to be a minimal subset of X such that $up(X) = up(\lfloor X \rfloor)$. We can then use $\lfloor X \rfloor$ to describe X sufficiently, as seen below. To check whether a graphs G_f is coverable using the set of rules $\mathcal{R}$, we compute the following sequence: $Cover_0 = \{G_f\}$, $Cover_{i+1} = \left\lfloor Cover_i \cup \bigcup_{r \in \mathcal{R}} Pre_r(Cover_i) \right\rfloor$. The sequence $Cover_0, Cover_1, \ldots$ resembles an exploration which starts from G_f and for each rule r, it applies the rule backwards to the current set of graphs by taking into account their denotation. The operator Pre_r is such that $G \in Pre_r(H)$ iff $\exists H'.\ H \sqsubseteq H',\ G \Rightarrow_r H'$. Hence in the case of subgraph inclusion, by adding auxiliary nodes and edges, we first expand H into graph H', and then we search for a possible match with the right-hand side R of rule r. If such a match exists, we apply a rewriting step backwards replacing R by L. The graph G_f is then coverable if there is an $H \in Cover_i$ with $H \sqsubseteq G_0$ for some $i \in \mathbb{N}_0$.

It is sufficient to compute a minimal set $MPre_r(H)$ of graphs that denotes the infinite set $Pre_r(H)$, i.e. $MPre_r(H) = \lfloor Pre_r(H) \rfloor$. In the case of subgraph inclusion this can be done by considering only expanded graphs H' that are obtained by merging H with subgraphs of R. Indeed we observe that if r is applied backwards to a subgraph of H' that does not overlap with H, then the resulting graph will still contain H as a subgraph, and will never be added (it is subsumed and does not bring any new information).

For the case of the minor ordering we can use similar ideas for the backwards step (see also [17]).

In general the sequence $Cover_0, Cover_1, \ldots$ need not become stationary, hence the procedure need not terminate, but we can show that it terminates for our specific cases. The termination depends mainly on the fact that the subsumption relation is a wqo on the used set of graphs, but may also be affected by the rule set.

3.2 Context-free Graph Transformation Systems

A well-known subclass of graph transformation systems are context-free graph grammars or hyperedge replacement graph grammars [11], where the left-hand side of a rule consists of a single hyperedge and no nodes are deleted or fused. Such grammars are usually used as a language-generating device: labels are partitioned into terminal and non-terminal labels. Only graphs which have only terminal labels are elements of the language. Furthermore G_0 consists of a single edge with a non-terminal label (the axiom) and the same is true for all left-hand sides. Since the distinction between terminal and non-terminal labels does not play a fundamental role for decidability questions, we will drop it in the following.

▶ **Definition 7.** A set $\mathcal{R}$ of rewriting rules is called *context-free* if every rule $r \in \mathcal{R}$ with $r\colon L \rightharpoonup R$ satisfies the following restrictions:

- L has the form $L = (\{v_1, \dots, v_n\}, \{e\}, [e \mapsto v_1 \dots v_n], l_L^E)$, i.e., L consists of a single hyperedge, which is connected to a duplicate-free sequence of nodes.
- r is defined and injective on $v_1, \dots, v_n$. Furthermore r is undefined on e.

Although not context-free, the termination detection example in Section 2 contains context-free rules, e.g. the deactivation rule (3a), the creation rule (3c) and the rule generating a termination message (3f).

▶ **Proposition 8** ([11]). *The reachability problem for context-free graph transformation systems is* NP*-complete. More precisely: there are context-free grammars for which the membership problem is* NP*-complete (in the size of G_f).*

▶ **Proposition 9.** *The coverability problem for context-free graph transformation systems is decidable.*

Proof sketch. The decision procedure is based on the backward exploration algorithm in Section 3.1, where we instantiate the predicate "G subsumes H" by $G \sqsubseteq_s H$ (G is a subgraph of H). Indeed, the key idea is to use a graph H as a symbolic representation of its upward closure wrt. $\sqsubseteq_s$. For the case of context-free rules, termination is obtained by showing that the number of hyperedges occurring in new graphs added at each iteration never increases. From this property and since the arity of hyperedges is finite, we have that the state space we need to explore to search for a graph that subsumes G_0 is always finite (but potentially exponential in the input). More details can be found in [3]. ◀

3.3 Restrictions on Node Deletion and Creation

In many cases graph transformation systems can be viewed as a variant of Petri nets with additional structure. In this case the graph transformation system can be translated into a low-level Petri net and we obtain decidability results from decidability results for Petri nets, possibly with reset and transfer arcs. This is usually the case when we impose some restrictions on the number of nodes that are deleted and/or created, since the main feature that gives GTS more expressiveness than Petri nets is the creation of new nodes. We can obtain such a GTS from the example of Section 2 by deleting the rule which creates new processes and the lossy system rules. To model the lossy system rules transfer arcs are required. This section relies on the intuitions and results of [2], which spells out the connection between SPO and nets with reset arcs. The type of graph transformation systems that are more or less equivalent to Petri nets can be specified as follows. Intuively the rules do not allow node deletion, creation or fusion.

▶ **Proposition 10.** *Assume that the set $\mathcal{R}$ of rewriting rules satisfies the restriction that every rule morphism $r\colon L \rightharpoonup R$ is a bijection on nodes. Then the reachability problem and the coverability problem are decidable.*

Proof sketch. Let G be the initial graph. In this setting V_G remains unchanged by any graph rewriting step and only the edges may change. We construct a Petri net from the GTS to reduce reachability and coverability to reachability and coverability on Petri nets. The places of the Petri net are defined as $P = \{(\ell, s) \in \Lambda \times V_G^* \mid ar(\ell) = |s|\}$. A token in a place (ℓ, s) represents an edge e with $l(e) = \ell$ and $c(e) = s$. The initial graph can be transformed straightforwardly into a marking of the Petri net. The rewriting steps are then simulated by adding a set of transitions for each rule $r\colon L \rightharpoonup R$, taking into account every possible match of r to any possible graph with $|V_G|$ nodes. We achieve this by enumerating all possible mappings h of nodes of L to nodes of V_G, adding one transition for each. For each edge e of L the transition has one input place $(l(e), h(c(e)))$. Analogously the transition has one output place for every edge in R using the same mapping. Effectively a transition takes tokens out of places representing the edges deleted by r and adds tokens in places representing the edges created by r. Since coverability and reachability are decidable for P/T nets, we obtain the same for this variant of GTS. ◀

If we allow node fusion and node deletion and hence also deletion of adjacent dangling edges, but recreate the same number of nodes, we are equivalent in expressivity to Petri nets with transfer arcs.

▶ **Proposition 11.** *Assume that the set $\mathcal{R}$ of rewriting rules satisfies the restriction that for every rule (partial) morphism $r\colon L \rightharpoonup R$ we have $|V_L| = |V_R|$. Then the reachability problem is undecidable, but the coverability problem is decidable.*

Proof sketch. Since the number of nodes of a graph stays constant during rewriting, the encoding of GTS into Petri nets is similar to the proof of Proposition 10. In order to deal with partial morphisms (i.e. node deletion) and non-injective ones (i.e. node fusion), we introduce transitions with transfer arcs that can transfer all tokens contained in a given set of places into a specific place. Reset arcs [12] are a special case in which the transferred tokens are moved to a sink place. Node deletion and subsequent recreation can be simulated via reset arcs, which empty all places corresponding to edges adjacent to a deleted node. Similarly, node fusion can be simulated by transfer arcs which merge the contents of all places corresponding to edges adjacent to nodes that have the same image. Hence we can encode all GTS conforming to the restrictions into transfer nets, inheriting the decidability result from coverability of transfer nets. On the other hand, every reset net can be encoded into a GTS with the above restrictions (see [2]). Hence reachability is undecidable for this class of GTS. ◀

3.4 Non-Deleting Graph Transformation Systems

Now consider GTS that are *non-deleting* (neither edges nor nodes) and in addition label-preserving, i.e., the rule morphism $r\colon L \rightarrowtail R$ is a (total) label-preserving injection. For instance rule (3d) in Section 2 is non-deleting.

▶ **Proposition 12.** *The reachability problem is decidable for non-deleting graph transformation systems.*

Proof sketch. In this setting reachability is decidable, due to the monotonicity of the rules. We apply all rules to derive all possible graphs and stop the derivation if a graph larger than G_f is reached. ◀

However the monotonicity has no effect on coverability and we will show that coverability is in fact undecidable.

▶ **Proposition 13.** *The coverability problem is undecidable for non-deleting graph transformation systems.*

Proof sketch. The coverability problem is undecidable for general GTSs because the halting problem for Turing machines can be reduced to it. This reduction is still possible for non-deleting GTSs. We use a directed path of labeled edges to represent the tape of the Turing machine and add one edge to mark the current state and head position. The input word forms the initial graph and non-deterministically additional blanks are added at both tape ends. Then for each step of the Turing machine, non-deleting rules copy the old tape with the appropriate changes and connect it to the old tape, such that the full Turing machine computation results in a grid-like graph. We have to take into account that without application conditions any rule can be applied an arbitrary number of times using the same matching, leading to a "branching" in the grid. However, we can show that this is not a problem, because different branchings do not interact. Hence, the Turing machine terminates if and only if an edge labeled with a final state is coverable in the GTS. The full proof can be found in [3]. ◀

3.5 Well-Structured Graph Transformation Systems

A good source for decidability results for the coverability problem are well-structured transition systems [16, 1]. For this we need a well-quasi order $\sqsubseteq$, possibly not on the class of all graphs, but on restricted classes. Furthermore it has to be shown that the well-quasi order is a (weak) simulation for the reduction relation, i.e., if $G \sqsubseteq H$ and G is rewritten to G', then H can be rewritten (possibly in several steps) to H' such that $G' \sqsubseteq H'$.

These conditions ensure that every upward-closed set can be finitely represented and that the set of predecessors of an upward-closed set is also upward-closed. If in addition the order is decidable and the direct predecessors can be effectively computed, one automatically obtains a backward search algorithm which can decide coverability.

Here we reuse results of [20, 17] and adapt them to our present setting (hypergraphs and SPO with injective matches as a rewriting formalism). The following decidability results then hold.

▶ **Proposition 14.** *If the set of rules contains edge contraction rules for each edge label (i.e., a rule deleting that edge and merging some of its adjacent nodes using any partition on the node set), then the coverability problem is decidable.*

Proof sketch. We prove well-structuredness of GTS with contraction rules for all possible edge labels. We first recall that the graph minor ordering is a decidable well-quasi ordering [22]. Furthermore, the transition system of a GTS with contraction rules is monotone wrt. the minor ordering. Based on these properties, we can apply the symbolic backward reachability algorithm in Section 3.1, by instantiating the *subsumes* relation with the minor relation, and by taking as $MPre_r$ the computation of the predecessors described in [17] for conflict-free matches, adapted here to injective matches. The fact that the minor ordering is a wqo ensures termination of the predecessor computation and thus of the entire decision procedure. Note also that coverability with subgraph inclusion can be easily reduced to coverability wrt. minor ordering by adding a rule that detects the subgraph G_f and adds an edge with a unused label. Then we check whether this new edge is the minor of a reachable graph. ◀

The GTS of Example 4 satisfies the conditions of Proposition 14 because of the lossy system rules (Figure 4). Hence the presented backward search can be used to automatically show that the protocol is erroneous, as shown in Figure 7. Beginning with the unwanted pattern, i.e. an active process and the termination flag, the rules are applied backwards until ultimately reaching a minor of the initial graph. In fact the result is a set of minimal graphs describing all initial graphs for which the protocol is erroneous. Note that in the first step the graph first has to be extended, adding the passive detector, before the rule can be applied backwards. From the resulting sequence of rule applications it is apparent that the

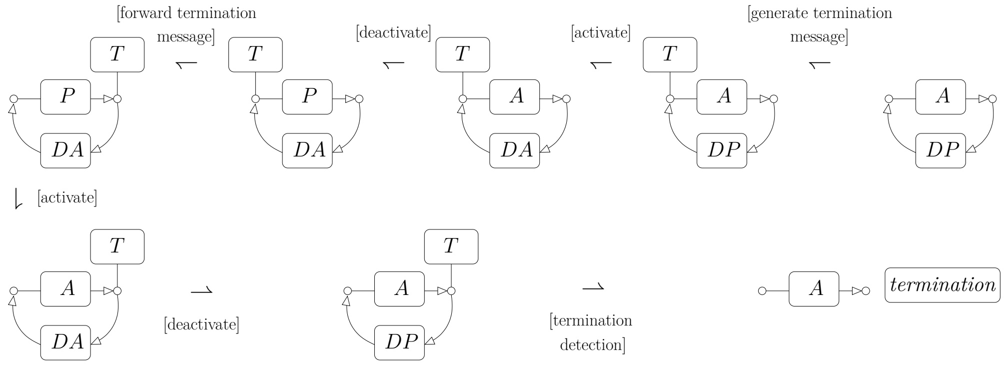


Figure 7 Illustration of a backward search using the GTS of Example 4

protocols error occurs because a passive process which forwarded the termination message can be activated afterwards. Hence the termination message and the process activation zone both move around the ring, but never meet.

▶ **Definition 15.** For a hypergraph G a *path* of length n is a sequence $v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n$ of nodes $v_i \in V_G$ and edges $e_i \in E_G$, where e_i is adjacent to v_{i-1}, v_i and no node or edge occurs more than once. A n-bounded path graph is a hypergraph G in which all paths have length less than or equal to n.

The following proposition is inspired by results in [20].

▶ **Proposition 16.** *If it is known that every graph reachable from G_0 is an n-bounded path graph, then the coverability problem is decidable for the given GTS.*

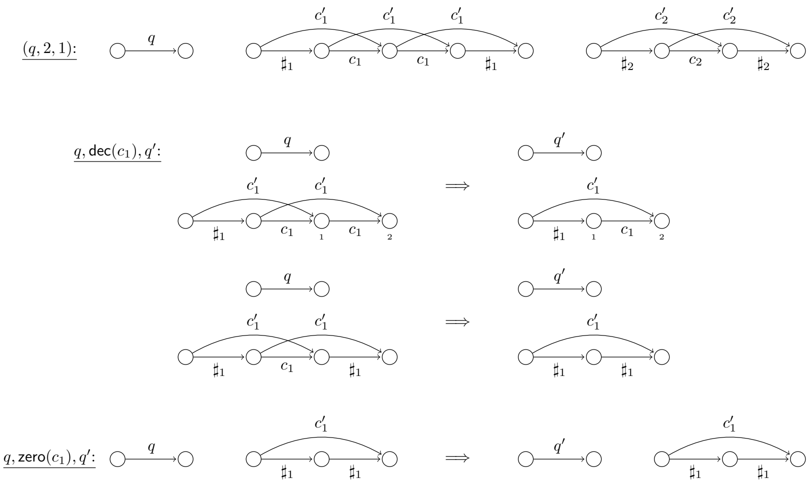
Proof sketch. We prove well-structuredness of GTS wrt. subgraph ordering in the class of n-bounded path graphs. We can then apply the symbolic backward reachability algorithm in Section 3.1 by instantiating the subsumption relation $\sqsubseteq$ with subgraph inclusion, and by discarding all predecessors that do not satisfy the n-bounded path property. Termination is guaranteed here by the fact that subgraph ordering for bounded path graphs is a wqo [10]. ◀

3.6 Graph Transformation with Deletion by Minor Rules

The class of graph transformation systems with minor rules (node/edge deletion, edge contraction) is interesting in its own right. One can allow any combination of those rules and study decidability questions.

A direct consequence of the decidability of the coverability problem (wrt. the minor ordering) for graph transformation systems with edge contraction rules [17] is the decidability of the reachability problem for the class of graph transformation systems that contain node deletion, edge contraction and edge deletion rules. In fact, for these kind of systems, reachability can be reduced to coverability because all the rules which are used in the minor ordering are also present in the system. Here we investigate what happens to the reachability problem when considering graph transformation systems where not all three minor rules are present. In the sequel, we will say that a graph transformation system is:

- *edge-contracting* if the set of rules contains all edge contraction rules for each edge label, excluding the edge deletion rule which is a special case of edge contraction;



■ **Figure 8** Encoding of a Minsky machine

- *edge-deleting* if the set of rules contains edge deletion rules for each edge label;
- *node-deleting* if the set of rules contains a node deletion rule.

We have the following result which shows that to obtain the decidability of the reachability problem in graph transformation systems with minor rules, all three types of minor rules are necessary.

▶ **Proposition 17.** *The reachability problem is undecidable for the following classes of graph transformation systems: (a) edge-deleting and node-deleting; (b) edge-contracting and node-deleting; (c) edge-deleting and edge-contracting.*

Proof sketch. We encode reachability for two counter machines. A two counter machine consists of a finite set of instructions manipulating two counters c_1 and c_2. Instructions have the following form (the semantics is the intuitive one): $(q, \mathsf{inc}(c_i), q')$ (increment c_i), $(q, \mathsf{dec}(c_i), q')$ (decrement c_i), $(q, \mathsf{zero}(c_i), q')$ (a blocking zero-test on c_i), where q, q' are control states. Given a (q, k, l) with $k, l \in \mathbb{N}$, it is undecidable to determine whether the program can reach the configuration (q', k', l') starting from $(q_0, 0, 0)$.

A configuration like $(q, 2, 1)$ is encoded as a path with additional crossing edges as shown Fig. 8. Such an encoding of configurations ensures that deletion by minor rules either blocks a simulation or introduces some garbage that cannot be removed. In both cases we define a reachability query that ensures that the simulation will not take wrong turns. Decrement and increment are encoded via graph transformations that preserve the structure of the representation as in Fig. 8. The correctness of the reduction is discussed in [3]. ◀

▶ **Proposition 18.** *The coverability problem is undecidable for graph transformation systems with edge deletion and node deletion rules.*

Proof sketch. The proof is a variant of the proof of Proposition 17 in the case of edge deletion and node deletion. More details can be found in [3]. ◀

3.7 Relabelling Rules

We now consider rules with arbitrary left-hand sides that are however only allowed to relabel their nodes or edges. So far, node labels were of no specific interest, but they play an important role in this section. Hence we now generalize the notion of hypergraph to include node labels: to a quadruple of the form $G = (V_G, E_G, c_G, l_G^E)$ representing a graph as introduced in Definition 1, we will now add an additional node-labelling function $l_G^V\colon V_G \to \Lambda'$, where Λ' is a finite set of node labels. The notion of graph morphism and the notion of rewriting are extended in the obvious way.

▶ **Definition 19.** A rewriting rule $r\colon L \rightharpoonup R$ is called a *relabelling rule* if r is a bijective (but not necessarily label-preserving) morphism. A node or edge x is said to be relabelled if $l_L(x) \neq l_R(r(x))$.

Example 4 can be seen as a relabelling system when deleting the lossy system rules as well as the rule creating new processes. The termination message and flag can be realised using two node labels or two edge labels where one indicates the existence and one the non-existence of the message and flag respectively.

Since reachability and coverability from a fixed initial graph are clearly decidable in this setting (the set of derivable graphs is finite), we consider the following *existential coverability* problem: assume that we are given a set of rewriting rules $\mathcal{R}$, a set of initial labels and a final graph G_f. Is there a graph G_0 labelled with only initial labels and a graph H such that $G_0 \Rightarrow_{\mathcal{R}}^* H$ and G_f is a subgraph of H?

The existential coverability is of interest when analysing distributed systems. By modelling an algorithm for a distributed system using a GTS, one effectively obtains a relabelling GTS because the system's topology remains unchanged during execution of the algorithm. When G_f represents an error configuration, the existential coverability problem transforms to the question: is there a distributed system where the modelled algorithm produces the specified error? A slightly different approach using node labels is pursued in [6], where the set of labels may be infinite and problems more specific for distributed systems are studied.

We first consider GTS with either only edge relabelling or only node relabelling rules, i.e. either the set of node labels or the set of edge labels is a singleton.

▶ **Proposition 20** (Edge Relabelling)**.** *The existential coverability problem is decidable for graph transformation systems with relabelling rules where the set of node labels Λ' is a singleton, i.e., $|\Lambda'| = 1$.*

Proof sketch. Coverability can be decided by a simple fixed-point computation which determines the set of "reachable" edge labels. A label is reachable if it is initial or occurs on a right-hand side of some rule, where all labels of the corresponding left-hand side are reachable. Then the graph G_f is coverable if and only if its edges are labelled with only reachable labels.

Assume all labels of G_f are reachable. Then for every edge e_i of G_f there is an initial graph G_{e_i} and a sequence of relabelling steps leading to some graph covering the edge. By taking all graphs G_{e_i} and merging appropriate nodes, an initial graph can be obtained from which G_f is coverable by combining the relabelling steps used to cover the individual edges. The nodes must thereby be merged such that the covered edges are connected to each other in a proper way to form G_f after relabelling. It can also be shown that G_f is not coverable if it contains a non-reachable label. The full proof can be found in [3]. ◀

▶ **Proposition 21** (Node Relabelling)**.** *The existential coverability problem is decidable for graph transformation systems with relabelling rules where the set of edge labels Λ is a singleton, i.e., $|\Lambda| = 1$.*

Proof sketch. The proof for this proposition is analogous to the proof of Proposition 20 using node labels instead of edge labels. An initial graph for the whole graph can be constructed by taking one initial graph for each occurrence of a node label (covering that label) and adding every possible edge (i.e., generating a complete graph). ◀

Now assume there are both node and edge labels and both can be modified, then the existential coverability problem is undecidable. The graph structure cannot be disregarded in this case and therefore the main assumption of the proofs above does not hold.

▶ **Proposition 22** (Node and Edge Relabelling)**.** *The existential coverability problem is undecidable for graph transformation systems with (general) node and edge relabelling rules.*

Proof sketch. A GTS can be constructed, which simulates a Turing machine. It first extracts a path out of the initial graph and then simulates a Turing machine with the path as tape, where each edge label is a cell of the tape. The node labels are thereby used to ensure that the extracted structure is actually a path, i.e. they ensure that at most two edges attached to the node belong to the tape. The Turing machine computation terminates if and only if there is a sufficiently large initial graph, containing a large enough tape and resulting in a graph covering the final state. The full proof can be found in [3]. ◀

Hence we arrive at the surprising conclusion that while for edge or node relabelling only their existential coverability is easily decidable, their combination results in an undecidable problem. The fact that graphs with edge and node labels can be encoded into graphs with edge labels does not help, since the encoding is not surjective and the corresponding relabelling problems cannot be reduced to each other: there could always be a graph outside of the image of the encoding from which we can cover the given subgraph.

4 Related Work

In this paper we focused our attention on single-pushout (SPO) rewriting. Another possibility would be to use double-pushout (DPO) rewriting [7]. In DPO a node cannot be deleted if it is connected to edges that are not explicitly deleted. The relation between Petri nets and DPO GTS has been studied in [2], where the encoding of nets into GTS deletes and recreates nodes in order to simulate the effects of inhibitor arcs from which we get undecidability of reachability and coverability even for rules that maintain the number of nodes always constant.

Well-structuredness of concurrency models in the class of bounded path graphs has been considered e.g. in [20, 23]. In all above mentioned models reduction rules have a restricted form to model either rendezvous or broadcast communication. In this paper we generalize well-structuredness to reduction rules defined by total injective morphisms. Well-structured graph rewriting is also considered in [9] where reduction rules that involve all neighbours (independently from their actual number) of a node are used to model broadcast communication. This type of rules cannot be modelled via GTS. Extending the language so as to capture broadcast communication is a possible future research direction.

Decidability boundaries for reachability problems of fragments of a graph-based model of biological systems are given in [8]. In κ a configuration consists of a graph in which nodes have labels and have a fixed number of binding sites. Rules can test and update node labels and the presence or absence of a binding site. Undecidability of coverability in GTS without deletion is inspired by a similar result for κ. The proof for GTS however does not require (to test on) node labels: it is uniquely based on an increasing graph structure used to simulate the evolution of an unbounded tape of a Turing machine. Furthermore, we consider here several other classes of reduction rules (e.g. context-free, minor and bounded path, relabeling) that are not studied in [8].

5 Conclusion

The results concerning decidability of reachability and coverability (excluding the relabelling cases) can be summarised in the diagram in Fig. 9. Interestingly, for some classes of GTS the reachability problem is decidable and the coverability problem undecidable, while for other classes it is the other way round. Of particular interest is the case of non-deleting GTS, into which we can encode Turing machines, however without guaranteeing termination for halting machines. Hence these GTS cannot be considered Turing-complete in the sense discussed in [19].

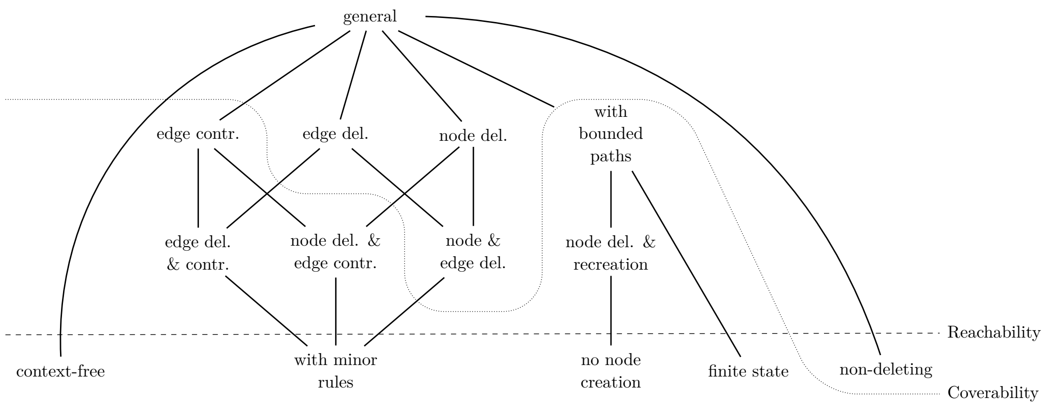


Figure 9 Decidability and Undecidability Boundaries.

We have obtained very general (un)decidability results that can be applied to all kinds of dynamic systems with evolving topologies. They can serve as a general toolbox to obtain decidability results for process calculi, which can often be straightforwardly encoded into graph transformation, for biological systems and for other formalisms. Note that, although we used hypergraphs, our undecidability proofs are formulated such that they also hold for directed multigraphs.

The studied subclasses of GTS can also be found in practice: first, in many examples in the literature the system is indeed fairly static and the number of nodes is fixed. Still, GTS here have a modelling advantage over Petri nets. Furthermore the node/edge deletion and edge contraction rules can be used to faithfully model lossy systems. Finally, triple graph grammars, specifying model transformations, usually have non-deleting rules.

Several of the decision procedures listed in this paper, especially those for coverability, can be implemented in practice and have reasonable runtimes. For instance, there are implementations of the coverability algorithm for Petri nets and of the backwards search algorithm described in [17].

One unsolved problem remains: the exact complexity of coverability for context-free GTS. We currently know that the problem is in PSPACE and we have an NP-hardness proof, but the exact complexity is open. We hope that this paper will stimulate further research in this area and lead to a better understanding of the algorithmic aspects of graph transformation systems.

Acknowledgements: We would like to thank Roland Meyer for interesting discussions about several aspects of this work.

References

1 P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS '96*, pages 313–321. IEEE, 1996.

2 P. Baldan, A. Corradini, and U. Montanari. Relating SPO and DPO graph rewriting with petri nets having read, inhibitor and reset arcs. In *Proc. of PNGT '04*, volume 127.2 of *ENTCS*, pages 5–28, 2005.

3 Nathalie Bertrand, Giorgio Delzanno, Barbara König, Arnaud Sangnier, and Jan Stückrath. On the decidability status of reachability and coverability in graph transformation systems. Technical Report DISI-TR-11-04, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, 2012.

4 F. Bonchi, F. Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *Proc. of FOSSACS '09*, pages 272–287. Springer, 2009. LNCS 5504.

5 N. Busi and G. Zavattaro. Deciding reachability in mobile ambients. In *Proc. of ESOP '05*, pages 248–262. Springer, 2005. LNCS 3444.

6 J. Chalopin, Y. Métivier, and W. Zielonka. Election, naming and cellular edge local computations. In *Proc. of ICGT '04 (International Conference on Graph Transformation)*, pages 242–256. Springer, 2004. LNCS 3256.

7 A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation—part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, chapter 3. World Scientific, 1997.

8 G. Delzanno, C. Di Giusto, M. Gabbrielli, C. Laneve, and G. Zavattaro. The *kappa*-lattice: Decidability boundaries for qualitative analysis in biological languages. In *Proc. of CMSB '09*, pages 158–172. Springer, 2009. LNCS 5688.

9 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *Proc. of CONCUR '10*, pages 313–327. Springer, 2010. LNCS 6269.

10 G. Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992.

11 F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, chapter 2. World Scientific, 1997.

12 C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP '98*, pages 103–115. Springer, 1998. LNCS 1443.

13 H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation—part II: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, chapter 4. World Scientific, 1997.

14 C. Ermel, H. Ehrig, F. Orejas, and G. Taentzer, editors. *Proc. of GraMoT '10*, volume 30 of *Electronic Communications of the EASST*, 2010.

15 J. Esparza and M. Nielsen. Decidability issues for Petri nets. Technical Report RS-94-8, BRICS, May 1994.

16 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

17 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *Proc. of CAV '08*, pages 214–226. Springer, 2008. LNCS 5123.

18 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. Technical report, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, 2012. Corrected version, available from `http://duepublico.uni-duisburg-essen.de/go/technische-berichte`.

19 S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. *Theoretical Computer Science*, 330:501–551, 2005.

20 R. Meyer. *Structural Stationarity in the π-Calculus.* PhD thesis, Carl-von-Ossietzky-Universität Oldenburg, 2009.

21 A. Rensink. The GROOVE simulator: A tool for state space generation. In *Proc. of AGTIVE '03*, pages 479–485. Springer, 2003. LNCS 3062.

22 Neil Robertson and Paul Seymour. Graph minors XXIII. Nash-Williams' immersion conjecture. *Journal of Combinatorial Theory Series B*, 100:181–205, March 2010.

23 T. Wies, D. Zufferey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *Proc. of FOSSACS '10*, pages 94–108. Springer, 2010. LNCS 6014.