UNIVERSITÄT
DUISBURG
ESSEN

INFORMATIK

ABTEILUNG FÜR INFORMATIK UND
ANGEWANDTE KOGNITIONSWISSENSCHAFT

FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN

Technischer Bericht Nr. 2012-03

Well-Structured Graph Transformation Systems with Negative Application Conditions

Barbara König
Jan Stückrath

3.7.2012

ISSN 1863-8554

IMPRESSUM:

Technische Berichte der Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen

ISSN 1863-8554

Herausgeber:

Abteilung für Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen
Campus Duisburg
47048 Duisburg

http://duepublico.uni-duisburg-essen.de/informatik/berichte.xml

Well-Structured Graph Transformation Systems with Negative Application Conditions[*]

Barbara König and Jan Stückrath

Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany

Abstract. Given a transition system and a partial order on its states, the coverability problem is the question to decide whether a state can be reached that is larger than some given state. For graphs, a typical such partial order is the minor ordering, which allows to specify "bad graphs" as those graphs having a given graph as a minor. Well-structuredness of the transition system enables a finite representation of upward-closed sets and gives rise to a backward search algorithm for deciding coverability. It is known that graph tranformation systems without negative application conditions form well-structured transition systems (WSTS) if the minor ordering is used and certain condition on the rules are satisfied. We study graph transformation systems with negative application conditions and show under which conditions they are well-structured and are hence amenable to a backwards search decision procedure for checking coverability.

1 Introduction

Graph transformation systems (GTS) [16] are a Turing-complete model of computation, which means that many properties of interest, especially concerning reachability and coverability ("Is it possible to reach a graph that contains a given graph as a subgraph?") are undecidable. Naturally, one obtains decidability of both problems when restricting to finite-state graph transformation systems, i.e., systems where only finitely many graphs up to isomorphism are reachable from a given start graph. However, similar to the case of Petri nets [5], it is possible to obtain decidability results also for certain (restricted) classes of infinite-state graph transformation systems [2]. This is important for many applications, since systems with infinitely many states arise easily in practice.

A good source of decidability results for the coverability problem are so-called well-structured transition systems [7,1]. They consist of a (usually infinite) set of states, together with a transition relation and a well-quasi-order (see Definition 1), such that the well-quasi-order is a simulation relation for the transition system. Standard place/transition nets can be seen as well-structured transition systems, furthermore systems with some degree of lossiness (such as lossy channel systems, where channels might lose messages) are well-structured.

[*] Research partially supported by DFG project GaReV.

Well-structuredness implies that every upward-closed set of states can be represented by a finite set of minimal states (this is a direct consequence of the properties of a well-quasi-order). Under some additional conditions it is possible to perform a backwards search in order to compute and represent (via minimal states) all predecessors of an upward-closed set. This allows to answer coverability questions algorithmically.

In [11,10] we have shown how (single-pushout) graph transformation systems with edge contraction rules can be seen as well-structured transition systems. As well-quasi-order we used the minor ordering on graphs, which is shown to be a well-quasi-order in the famous Robertson-Seymour theorem [14,15].

However, the theory in [11] does not apply to graph transformation systems with negative application conditions [8,4], which often arise in practice. Such negative application conditions disallow the application of a rule if a certain "forbidden" subgraph is present.

Here we study such graph transformation systems with negative application conditions and show that they are well-structured under certain conditions (for instance in the presence of deletion and contraction rules that arise naturally in lossy systems). While this result is fairly straightforward to prove, it is more difficult to perform a backwards step and hence obtain a decision algorithm. We here give a general procedure for computing the predecessor set and show that it terminates in specific cases, i.e., for certain types of negative application conditions. We illustrate the theory with various examples, especially we study a (faulty) termination detection protocol and apply the decision procedure to the set of rewriting rules describing the protocol.

Proofs can be found in Appendix A.

2 Preliminaries

2.1 Well-Structured Transition Systems

We will now give the definitions concerning well-structured transition systems, following the presentation in [7].

Definition 1 (wqo and upward closure). *A quasi-order*[1] $\leq$ *(over some set X) is a* well-quasi-order *(wqo) if for any infinite sequence $x_0, x_1, x_2, \ldots$ of elements of X, there exist indices $i < j$ with $x_i \leq x_j$.*

An upward-closed set *is any set $I \subseteq X$ such that $x \leq y$ and $x \in I$ implies $y \in I$. A* downward-closed set *can be defined analogously.*

For a subset $Y \subseteq I$, we define its upward closure $\uparrow Y = \{x \mid \exists y \in Y : y \leq x\}$. Then, a basis *of an upward-closed set I is a set I^b such that $I = \uparrow I^b$.*

The definition of well-quasi-orders gives rise to some properties which are especially important for the backwards algorithm presented later.

Lemma 1. *Let $\leq$ be a well-quasi-order, then the following two statements hold:*

[1] Note that a quasi-order is the same as a preorder.

1. *Any upward-closed set I has a finite basis.*
2. *For any infinite, increasing sequence of upward-closed sets $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ there exists an index $k \in \mathbb{N}$ such that $I_i = I_{i+1}$ for all $i \geq k$.*

Definition 2 (Well-structured transition system). *A* well-structured transition system *(WSTS) is a transition system $T = (S, \Rightarrow, \leq)$, where S is a set of states and $\Rightarrow \subseteq S \times S$, such that the following conditions hold:*

1. ***Well-quasi-ordering:** $\leq$ is a well-quasi-order on S.*
2. ***Compatibility:** For all $s_1 \leq t_1$ and a transition $s_1 \Rightarrow s_2$, there exists a sequence $t_1 \Rightarrow^* t_2$ of transitions such that $s_2 \leq t_2$.*

$$\begin{tikzcd} t_1 \arrow[r, Rightarrow, "*"] \arrow[d, phantom, "\rotatebox{90}{$\leq$}"] & t_2 \arrow[d, phantom, "\rotatebox{90}{$\leq$}"] \\ s_1 \arrow[r, Rightarrow] & s_2 \end{tikzcd}$$

Given a set $I \subseteq S$ of states we denote by $\mathit{Pred}(I)$ the set of direct predecessors of I, i.e., $\mathit{Pred}(I) = \{s \in S \mid \exists s' \in I \colon s \Rightarrow s'\}$. Furthermore $\mathit{Pred}^*(I)$ is the set of all predecessors which can reach some state of I with an arbitrary sequence of transitions. Let $(S, \Rightarrow, \leq)$ be a WSTS. Backward reachability analysis involves the computation of $\mathit{Pred}^*(I)$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ where $I_0 = I$ and $I_{n+1} = I_n \cup \mathit{Pred}(I_n)$. However, in general this may not terminate. For WSTS, if I is upward-closed then it can be shown that $\mathit{Pred}^*(I)$ is also upward-closed (compatibility condition) and that termination is guaranteed (see Lemma 1).

Definition 3 (Effective pred-basis). *A WSTS has an* effective pred-basis *if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow\mathit{Pred}(\uparrow\{s\})$.*

Now assume that T is a WSTS with effective pred-basis. Pick a finite basis I^b of I and define a sequence $K_0, K_1, K_2, \ldots$ of sets with $K_0 = I^b$ and $K_{n+1} = K_n \cup pb(K_n)$. Let m be the first index such that $\uparrow K_m = \uparrow K_{m+1}$. Such an m must exist by Lemma 1 and we have $\uparrow K_m = \mathit{Pred}^*(I)$.

The *covering problem* is to decide, given two states s and t, whether starting from a state s it is possible to cover t, i.e. to reach a state t' such that $t' \geq t$. The decidability of the covering problem follows from the previous argument: we define $I = \uparrow\{t\}$ and check whether $s \in \mathit{Pred}^*(I)$, i.e., if there exists a $\overline{s} \in K_m$ such that $\overline{s} \leq s$.

Theorem 1 (Covering problem [7]). *The covering problem is decidable for a WSTS with an effective pred-basis and a decidable wqo $\leq$.*

Thus, if T is a WSTS satisfying the extra conditions of Theorem 1 and the "error states" can be represented as an upward-closed set I, then it is decidable whether any element of I is reachable from the start state.

2.2 Graph Transformation Systems

We will now introduce the necessary preliminaries in order to define single-pushout (SPO) graph rewriting. Note that the minor relation used in the following admits a characterization via partial graph morphisms and hence straightforwardly integrates with SPO, which is defined in the category of partial morphisms. The concatenation of a rule and a minor morphism is again a rule.

Definition 4 (Hypergraph). *Let Λ be a finite set of labels and a function $ar\colon \Lambda \to \mathbb{N}$ that assigns an* arity *to each label. A (Λ-)*hypergraph *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, $c_G\colon E_G \to V_G^*$ is a connection function and $l_G\colon E_G \to \Lambda$ is the labelling function for edges. We require that $|c_G(e)| = ar(l_G(e))$ for each edge $e \in E_G$.*

We will simply use *graph* to denote a hypergraph. To simplify the necessary computations we will only use hypergraphs with at most binary edges, i.e. $|ar(\ell))| \leq 2$ for all labels ℓ. However note that large parts of the theory (except Proposition 2) can be extended to the general case.

Definition 5 (Graph morphism). *Let G, G' be (Λ-)graphs. A* partial graph 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 $l_G(e) = l_{G'}(\varphi_E(e))$ and $\varphi_V(c_G(e)) = c_{G'}(\varphi_E(e))$ 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.* Total morphisms *are denoted by an arrow of the form $\to$.*

We will now introduce the notion of (SPO) graph rewriting ([12]) with negative application conditions.

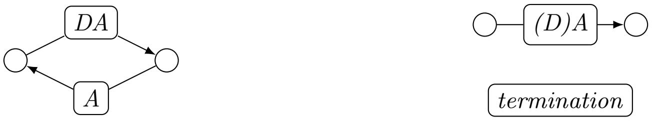
Definition 6 (Graph rewriting). *A* rewriting rule *is a partial morphism $r\colon L \rightharpoonup R$ together with a finite set of negative application conditions. A* negative application condition *(NAC) is a total, injective morphism $n_i\colon L \to N_i$. A* match *is a total, injective morphism $m\colon L \to G$. We say that a match m satisfies a NAC n_i if there is* no *total, injective morphism $n_i'\colon N_i \to G$ such that $n_i' \circ n_i = m$. A rule is* applicable *to a graph G if there is a match satisfying all NACs.*

For a pair of a rule r and a match m applicable to G, a rewriting step *is obtained by taking the pushout of m and r in the category of partial graph morphisms. Then G is rewritten to the pushout object H (written as $G \overset{r,m}{\Rightarrow} H$ or simply $G \Rightarrow H$).*

In [11] so called *conflict-free* matches are used, which may be non-injective. However, these matches cannot be used with our variant of negative application conditions, since the NAC and its match to G are injective and can therefore not commute with any non-injective match m. Note that any injective match is also conflict-free.

In this paper a graph transformation system (GTS) is simply a finite set of rules, not necessarily associated with an initial graph. For verification purposes initial and final graphs can be used as shown in the example below.

Example 1. For the later illustration of a backward step and the backward search algorithm, we introduce the following termination detection protocol as a running example. A similar protocol was used in [2] but without negative application



(a) A possible initial graph

(b) The final graph

Fig. 1: The protocols initial graph and it error configuration

conditions. For lossy systems this protocol is erroneous and we will show how the backwards search will detect the error.

The protocol consists of normal processes which can be active (A) or passive (P) and a detector process which can be active (DA) or passive (DP). The label $(D)A$ thereby stands both for an active detector and an active normal process ($(D)P$ is used analogously). The initial graph is a directed circle with an active detector and an active (normal) process (see Fig. 1a). Additional active processes can be generated (Fig. 2d) and active processes may become passive (Fig. 2b). When the detector becomes passive *start* and *end* flags are created and attached to the corresponding detector (Fig. 2a). The *end* flag can be forwarded along passive processes (Fig. 2e) and when the *end* flag was forwarded around the entire ring to reach the detector again, a *termination* flag is created (Fig. 2f) stating that all processes are passive. Any active process can reactivate a passive process (Fig. 2c) if there is no *start* flag between them. This ensures that all processes (including the detector) between the *start* and the *end* flag are passive. The absence of the *start* flag is thereby ensured by a negative application condition, which is indicated by the dashed edge.

Additionally there are rules simulating the lossiness of the system. Processes can leave the ring (Fig. 2g, Fig. 2h) and flags can be lost (Fig. 2i, Fig. 2j). We will later show that this GTS is well-structured.

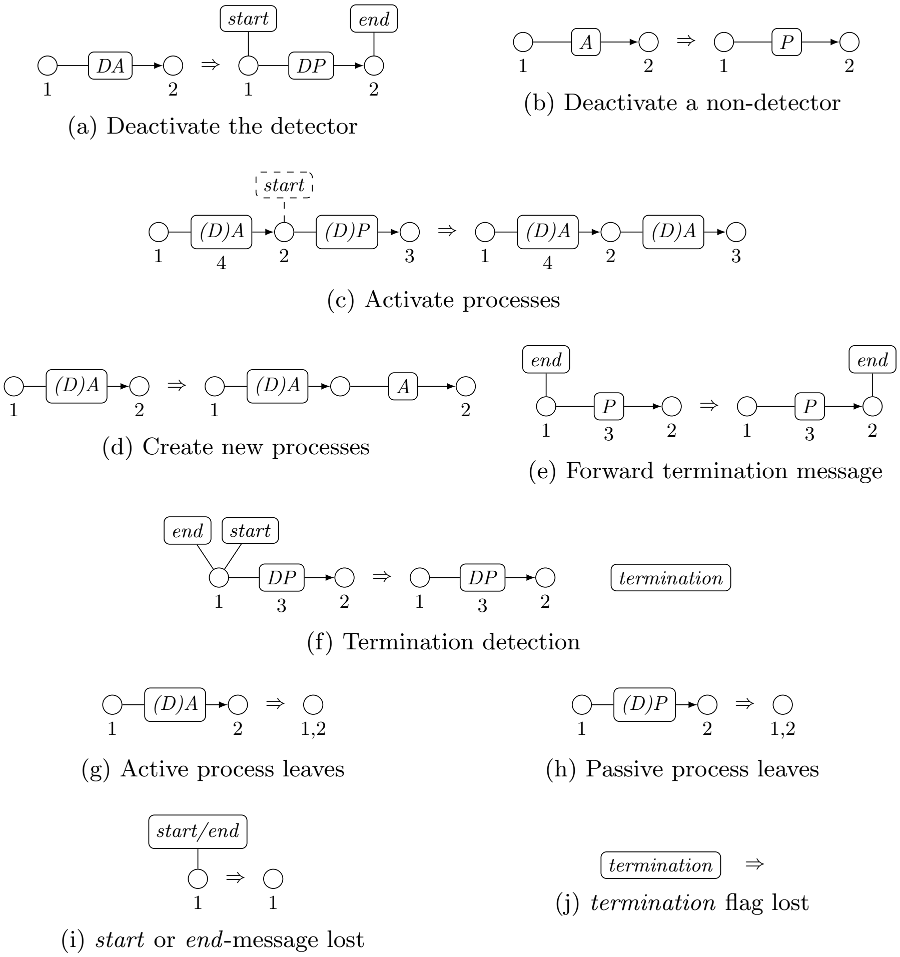
The protocol is correct if and only if from the initial graph no graph can be reached which contains the final graph (Fig. 1b), because this would mean that a *termination* flag was generated although there still exists an active process.

2.3 Minors and Minor Morphisms

We will now introduce the notion of graph minor [14,15] and recall some results from [11].

Definition 7 (Minor). *A graph M is a* minor *of a graph G (written $M \leq G$), if M can be obtained from G by (repeatedly) performing the following operations on G:*

1. *Contraction of an edge. In this case we remove the edge, choose an arbitrary partition on the nodes connected to the edge and merge the nodes as specified by the partition. (This includes edge deletion as a special case.)*
2. *Deletion of an isolated node.*



(a) Deactivate the detector

(b) Deactivate a non-detector

(c) Activate processes

(d) Create new processes

(e) Forward termination message

(f) Termination detection

(g) Active process leaves

(h) Passive process leaves

(i) *start* or *end*-message lost

(j) *termination* flag lost

Fig. 2: The basic rules of a termination detection protocol with NACs

Note that since we restrict the arity of any edge to at most two, there is only one possible partition if the arity is zero or one and only two possible partitions if the arity is two (one of which coincides with edge deletion).

The *Robertson-Seymour Theorem* [14] says that the minor order is a well-quasi-order even if the edges and vertices of the graphs are labelled from a well-quasi-ordered set, and also for hypergraphs and directed graphs (see [10,15]). Here we use the minor ordering presented in [11], but since we restrict the arity of edges, the ordering is essentially the same as for directed graphs.

Now any GTS with negative application conditions which satisfies the compatibility condition of Definition 2 (with respect to the minor ordering), can be analysed using the theory of WSTS. But before we characterize such GTS we first need the definition of minor morphisms and their properties. A *minor morphism* is a partial morphism that identifies a minor of a graph.

Definition 8 (Minor morphism). *A partial morphism $\mu : G \rightharpoonup M$ is a minor morphism (written $\mu : G \mapsto M$) if*

1. *it is surjective,*
2. *it is injective on edges* and
3. *whenever $\mu(v) = \mu(w) = z$ for some $v, w \in V_G$ and $z \in V_M$, there exists a path between v and w in G where all nodes on the path are mapped to z and μ is undefined on every edge on the path.*

For a minor morphism μ we define $\| \mu \|$ to be the number of nodes and edges on which μ is undefined.

We call a minor morphism μ a one-step *minor morphism if μ deletes exactly one node or contracts or deletes exactly one edge, i.e. additionally to the above restrictions it holds that either:*

1. *μ is injective, defined on all edges and defined on all but one node* or
2. *μ is defined on all nodes, defined on all but one edge and is injective on all nodes not attached to the undefined edge.*

In [15] a different way to characterize minors is proposed: a function, going in the opposite direction, mapping nodes of M to subgraphs of G. This however cannot be seen as a morphism in the sense of Definition 5 and we would have problems integrating it properly into the theory of graph rewriting. However, in [10] it is proven, that our minor ordering is a wqo and that the following facts about minor morphisms hold.

Lemma 2 ([11]). *The class of minor morphisms is closed under composition.*

Lemma 3 ([11]). *M is a minor of G iff there exists a minor morphism $\mu : G \mapsto M$.*

Lemma 4 ([11]). *Pushouts preserve minor morphisms in the following sense: If $f : G_0 \mapsto G_1$ is a minor morphism and $g : G_0 \to G_2$ is total, then the morphism f' in the pushout diagram below is a minor morphism.*

$$\begin{tikzcd}
G_0 \arrow[r, "f", |->] \arrow[d, "g"] & G_1 \arrow[d, "g'", harpoon] \\
G_2 \arrow[r, "f'", |->] & G_3
\end{tikzcd}$$

Note that Lemma 4 is also valid if g is not total and we require that f does not contract any edge deleted by g. In the following we will also use this variant of the lemma. Finally we need the following lemma, which is a weaker version of a related lemma in [11].

Lemma 5 ([11]). *Let $\psi_1 : L \to G$ be total and injective. If the diagram below on the left is a pushout and $\mu : H \mapsto S$ a minor morphism, then there exist minors M and X of R and G respectively, such that:*

1. the diagram below on the right commutes and the outer square is a pushout.
2. the morphisms $\mu_G \circ \psi_1 \colon L \to X$ and $\varphi_1 \colon M \to S$ are total.

$$\begin{tikzcd}
L \arrow[r, harpoon, "\psi_2"] \arrow[d, tail, "\psi_1"] & R \arrow[d, "\psi_1'"] & \\
G \arrow[r, harpoon, "\psi_2'"] & H \arrow[dr, mapsto, "\mu"] & \\
 & & S
\end{tikzcd}$$

$$\begin{tikzcd}
L \arrow[r, harpoon, "\psi_2"] \arrow[d, tail, "\psi_1"] & R \arrow[r, mapsto, "\mu_R"] \arrow[d, "\psi_1'"] & M \arrow[dd, "\varphi_1"] \\
G \arrow[r, harpoon, "\psi_2'"] \arrow[d, mapsto, "\mu_G"] & H \arrow[dr, mapsto, "\mu"] & \\
X \arrow[rr, harpoon, "\varphi_2"] & & S
\end{tikzcd}$$

Furthermore we need the following additional properties.

Lemma 6 (Reordering of minor decompositions). *Every minor morphism μ can be decomposed into a finite sequence of one-step minor morphisms μ_i such that $\mu = \mu_1 \circ \ldots \circ \mu_n$ and there are k, ℓ with $1 \leq k \leq \ell \leq n + 1$ and*

1. μ_i with $1 \leq i < k$ is undefined on one node (node deletion),
2. μ_i with $k \leq i < \ell$ is undefined on one edge and not injective on nodes (edge contraction) and
3. μ_i with $\ell \leq i < n + 1$ is undefined on one edge and injective on nodes (edge deletion).

3 Well-structuredness and Negative Application Conditions

One condition for the backwards algorithm to compute correct results is well-structuredness. In the following proposition we formulate a rather general condition for a GTS with negative application conditions which ensures that it is well-structured with respect to the minor ordering. We have to ensure that whenever $M \leq G$ and a rule $r \colon L \rightharpoonup R$ can be applied to M, the same rule can be applied to G. There are two problems that might disallow the rule application: G might contain a disconnected copy of the left-hand side L (which must be contracted to form a match of L in M) and G may contain more structure such that a valid match of L in M might not satisfy the NACs when extended to G. In both cases we ensure that G can be rewritten to G' in which both problems disappear.

Proposition 1. *A GTS containing rules with negative application conditions is well-structured wrt. the minor ordering if for every rule $r \colon L \rightharpoonup R$ with negative application conditions $n_i \colon L \to N_i$ and every minor morphism $\mu \colon G \mapsto M$ the following holds: for every match $m \colon L \to M$ satisfying all NACs (indicated by the crossed-out arrows), G can be rewritten to some graph G' such that there is a match $m' \colon L \to G'$ satisfying all NACs and a minor morphism $\mu' \colon G' \mapsto M$ with $m = \mu' \circ m'$ (see diagram below).*

$$\begin{tikzcd}
G \arrow[dr, |->, bend right, "\mu"'] & L \arrow[r, tail, "n_i"] \arrow[d, tail, "m"] & N_i \arrow[dl, tail, bend left, "/" marking] \\
 & M &
\end{tikzcd}
\qquad
\begin{tikzcd}
G \arrow[dr, Rightarrow, "*"] \arrow[ddr, |->, bend right, "\mu"'] & L \arrow[r, tail, "n_i"] \arrow[d, tail, "m'"] \arrow[dd, tail, bend left, "m"] & N_i \arrow[dl, tail, "/" marking] \arrow[ddl, tail, bend left, "/" marking] \\
 & G' \arrow[d, |->, "\mu'"] & \\
 & M &
\end{tikzcd}$$

"Lossy systems" (such as the GTS of Example 1) usually already satisfy Proposition 1 and other GTS can be transformed into GTS satisfying the conditions by adding rules for lossiness. In general this is possible by introducing rules that contract and deleted nodes and edges for every label (so called minor rules). The contraction rules enable the simulation of edge contractions performed by μ to generate a graph where the rule matches, while the deletion rules can be used to destroy matches of negative application condition. In the latter case it is sufficient to introduce rules which only delete edges that are associated with a negative application condition, such as rules $\nu_e \colon N \to N \setminus \{e\}$ for all edges e not in the range of $n \colon L \to N$. Note that any introduction of new rules will cause the backward search to compute an overapproximation for the original GTS.

For the computation of the finite basis of $\mathit{Pred}(S)$ we first introduce the notion of edge decontraction and bounding functions.

Definition 9 (Edge decontraction). *Let G be a graph. We define $\mathit{expand}(G)$ to be the set of all graphs $G' \geq G$ such that there is an edge e in G' and the contraction of e results in a graph isomorphic to G.*

Definition 10 (Bounding function). *Let $\mathcal{T}$ be a GTS with rule set $\mathcal{R}$ and let $b_r^{\mathcal{T}} \colon \mathit{Matches} \to \mathbb{N}_0$ be a function, where $(r \colon L \rightharpoonup R,\ \mathcal{N}) \in \mathcal{R}$ is a rule and Matches is the set of all matches $m \colon L \to G$ of r to some graph G, not necessarily satisfying all NACs $\mathcal{N}$. We call $b_r^{\mathcal{T}}$ a* bounding function *if every minor morphism $\mu \colon G' \mapsto G$, where there is a match $m' \colon L \to G'$ satisfying all NACs, can be decomposed into minor morphisms $\mu' \colon G' \mapsto M$ and $\mu'' \colon M \mapsto G$ satisfying the following properties:*

1. $\mu = \mu'' \circ \mu'$,
2. $\mu' \circ m'$ is a total, injective match of L in M and satisfies all NACs and
3. $\|\,\mu''\,\| \leq b_r^{\mathcal{T}}(m)$.

The bounding function is used to calculate the maximal number of decontractions needed to compute a graph of the predecessor basis. The existence of such a function guarantees termination of a backward step and we will prove its existence for special cases. In the following we will omit the superscript $\mathcal{T}$, since the GTS of interest is fixed.

We will now describe a procedure for computing an effective pred-basis (see Definition 3). In essence, given a graph S, we have to apply a rule $r \colon L \rightharpoonup R$ backwards. However, there are several complications, caused by the fact that S does not simply stand for itself, but represents a whole set of graphs, the upward-closure of S (i.e., all graphs that have S as a minor). Hence we have to consider the following facts:

- S might not contain an entire (co-)match of R, but it might represent graphs that contain such matches. In order to solve this problem we do not simply apply r backwards, but first compute all minors of R and look for matches of those minors (see Step 2 below).
- Whenever, after doing a backwards step, we find that the resulting graph X' contains a non-injective match of L, we do not discard X'. Again, this is because X' directly is not a predecessor, but it represents other graphs which are predecessors of graphs represented by S. Those graphs can be obtained by "forcing" the match to be injective via edge decontractions (Step 3).
- Finally, we have to solve the problem that the backwards step might result in a graph X which may contain an injective match, but does not satisfy the NACs. Similar to the case above, we have to find larger graphs represented by X by edge decontractions such that the resulting graphs do satisfy the NACs (Step 4).

The last item is more complex than the second-last: while we can bound the number of steps needed to "make" a match injective via decontractions by the number of nodes which are merged, this is not so easy in the case of NACs: by decontractions we may destroy a NAC, but create a new one. Since we want to represent upwards-closed sets wrt. a well-quasi-order, we can be sure that there are finitely many representatives. However, when searching for them, we might not know whether we have already found all of them. This is where the bounding function of Definition 2 comes into play in order to terminate the search.

Procedure 1. Let $\mathcal{T}$ be a GTS with the rule set $\mathcal{R}$ satisfying the compatibility condition, as described in Proposition 1. We assume that there is a bounding function for every rule of $\mathcal{T}$.

In the following we give a description of a procedure *pbn*(S) which generates a finite basis for the set of graphs reaching a graph larger or equal to S in one step. The first two steps are basically identical to the procedure presented in [11].

1. For each rule $(r\colon L \rightharpoonup R,\ \mathcal{N}) \in \mathcal{R}$, where $\mathcal{N}$ is the set of all negative application conditions of r, let $\mathcal{M}_{\mathcal{R}}$ be the set of all minor morphisms with source R. Furthermore let b_r be the bounding function of r.
2. For each $(\mu\colon R \mapsto M) \in \mathcal{M}_{\mathcal{R}}$ consider the rule $\mu \circ r\colon L \rightharpoonup M$ and perform the following steps.
3. For each total match $m''\colon M \to S$ compute all minimal pushout complements X' such that $m'\colon L \to X'$ below is total and injective on edges.[2] Then repeatedly apply the *expand*-function in order to split non-injectively matched nodes and to obtain a basis for all graphs X, of which X' is a minor and which contain an injective match of L (see diagram below).[3]

[2] The term *minimal* is used with respect to the minor ordering. For more details see [11].

[3] Alternatively the graph can be partially enlarged before applying the rule backwards to create an initial, injective match as explained in [2].

$$\begin{tikzcd}
 & L \arrow[r, "\mu \circ r"] \arrow[dl, "m"'] \arrow[d, "m'"] & M \arrow[d, "m''"] \\
X \arrow[r, mapsto] & X' \arrow[r, "r'"] & S
\end{tikzcd}$$

4. For each such X compute every total, injective morphism $n_i^j \colon N_i \to X$ of any NAC $(n_i \colon L \to N_i) \in \mathcal{N}$ that commutes with the match. If there is none, store X and proceed with the next X.

   If there is at least one such morphism, compute *expand*(X). For each $\overline{X} \in$ *expand*(X) do the following:
   - Let $\overline{\mu} \colon \overline{X} \mapsto X$ be the minor morphism that exists by Definition 9 and Lemma 3.
   - If there is no $\overline{m} \colon L \to \overline{X}$, such that $m = \overline{\mu} \circ \overline{m}$, i.e. the decontraction destroyed the match, discard X.
   - If there is a match $\overline{m}$ satisfying all NACs, store $\overline{X}$ and proceed with the next $\overline{X}$.

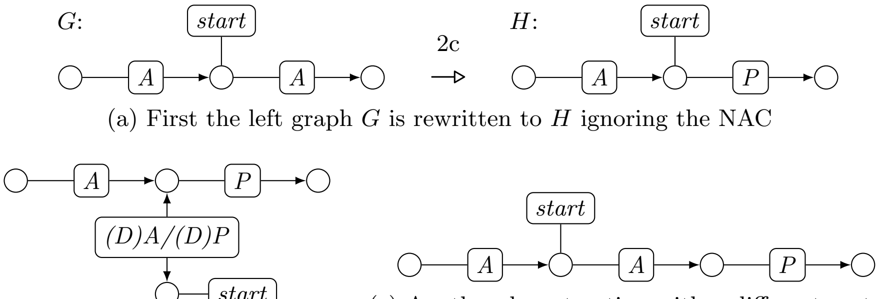
   If none of the previous two conditions hold, repeat Step 4 with the graph $\overline{X}$ and the match $\overline{m}$. Stop the recursion after $b_r(m)$ steps.[4]
5. The set *pbn*(S) contains all graphs X stored after the last repetition of the previous step.

Example 2. To illustrate the handling of negative application conditions we exemplarily apply rule 2c of our running example backward to the graph G in Figure 3a ($\longrightarrow\!\!\!\triangleright$ indicates a backward step). In Step 3 of Procedure 1 among others the graph H in Figure 3a is generated (μ is the identity), but the rule cannot be applied (in forward direction) to H to reach G since the negative application condition is not satisfied. Hence, in Step 4 the minimal set of graphs larger than H has to be found which can be rewritten to some graph larger than G. All eight resulting graphs are shown in Figure 3b. The "decontracted" edge in the middle can point in both directions and be labeled A, P, *DA* or *DP*. All other decontractions either destroy the match or do not destroy the match of the NAC, which in both cases produces graphs where the rule is not applicable.

Additional graphs will be generated if different matches of the left-hand side of rule 2c are used. For instance the graph in Figure 3c is generated (among others) if only one of the A-edges is matched to the right A-edge of G. An additional A-edge is created while generating an injective match. Note that the last graph is immediately deleted since it is larger than G and therefore already represented. A more comprehensive demonstration of backward steps is done in Section 4.

We will now state our main theorems: especially we will show that every graph generated by *pbn*(S) is in the predecessor set of the upwards-closure of S and vice versa.

[4] Note that applicability of the rule $\mu \circ r$ does not depend on μ and hence we do not need a bounding function $b_{\mu \circ r}$.



(a) First the left graph G is rewritten to H ignoring the NAC

(b) Eight possible decontractions

(c) Another decontraction with a different match

Fig. 3: Backward step of rule 2c containing a negative application condition

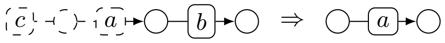


Fig. 4: A simple rule with a NAC (the dashed part)

Theorem 2. *The procedure pbn(S) (Procedure 1) computes a finite subset of Pred($\uparrow S$).*

Theorem 3. *If there is a bounding function b_r for every rule r, the set generated by pbn(S) is a finite basis of $\uparrow$Pred($\uparrow S$).*

Theorem 3 depends on the existence of a bounding function. As we will illustrate later, bounding functions are non-trivial to obtain, but we have bounding functions for specific negative application conditions.

Proposition 2. *There is a bounding function for all rules if the corresponding GTS satisfies the following properties:*

1. *$ar(\ell) \leq 2$ for all labels $\ell \in \Lambda$* and
2. *for every negative application condition $n\colon L \to N$ the graph N has at most one edge and two nodes in addition to $n(L)$.*

Example 3. In the general case the complexity of the bounding function can grow with the complexity of the match and the NAC. In particular there are rather simple NACs where decontractions cannot be discarded if they generate new matches of a NAC. Figure 4 shows a rule together with a NAC (dashed part of the graph). Assume the left graph in Figure 5 was generated by step 3 of the procedure. The two displayed decontractions lead to the right graph, but the middle graph does not satisfy the NAC and there is no single decontraction where the NAC is satisfied and the rightmost graph is represented.

Negative application conditions for arbitrary hypergraphs can therefore require a complex bounding functions.

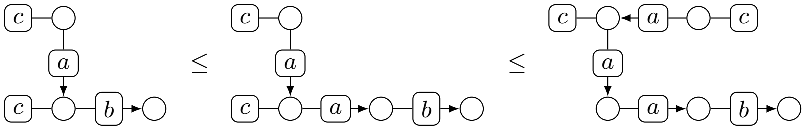


Fig. 5: Graphs generated from left to right via decontractions

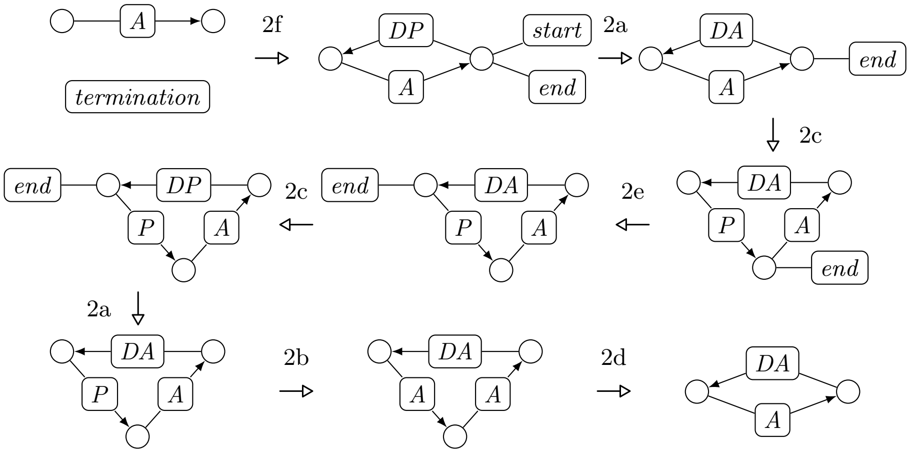


Fig. 6: An exemplary sequence of backward steps

4 Verification of a Termination Detection Protocol

As already mentioned the termination detection protocol presented in Example 1 is erroneous for lossy systems. Note also that its NAC satisfies the restrictions of Proposition 2 and can hence be analyzed automatically. One possibility to derive (a minor of) the initial graph from the final graph is shown in Figure 6. This derivation is found by the backward search within eight backward steps each indicated by $\multimap$ (together with the used rule). The rules used in the first three steps and the sixth step only partially match the graph, i.e. not the corresponding rule r is applied backwards, but the rule $\mu \circ r$ for some minor morphism μ (see Step 2 of Procedure 1). For instance the minor morphism μ used in the first step is injective and only undefined on the *DP*-edge, hence this edge is added when the rule is applied backwards. In the second step the minor morphism is undefined for the *end* flag, which is non-existent at the right place in the graph, but it is not added, since the rule creates it when applied forward. In all other steps (except the sixth) r is applied backwards directly, i.e. μ is the identity.

The most interesting step is the third, because rule 2c cannot be applied directly to the graph, since the match would be non-injective, but the rule can be applied to some larger graphs which are represented by the given graph. This backward step is shown in detail in Figure 7, where the diagram of Step 3 of the procedure is shown with the generated graphs. First the rule is applied backwards with a non-injective match replacing the A-edge with a P-edge, producing

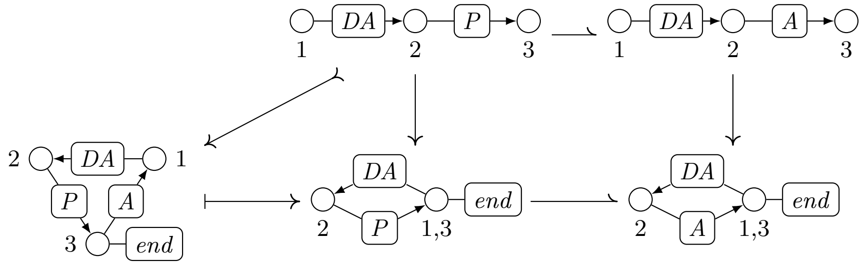


Fig. 7: The third backward step of the sequence in Fig. 6 in detail

another non-injective match. Since this is not a valid match in the sense of Definition 6, the rule is not directly applicable and the node 1,3 has to be split such that the match is injective. This is done by decontraction generating the A-edge (which is just one possibility). The resulting graph is one of the graphs represented by the previous graph, with the difference that the rule is applicable.

The other five steps are straightforward or similar to the first three. Note that since in the last step a graph is reached which is smaller or equal to (in this case isomorphic) the initial graph, the final state is coverable and the protocol erroneous.

Note that, seen in the other direction, this is not quite a sequence of rewriting steps of the GTS. Rather, by applying the rules in the forward direction, we sometimes obtain a graph different from the one in the figure, but represented by it. Still, following the sequence against the direction of the arrows, it is possible to reconstruct why the error occurs in the protocol: after the detector first turns passive, the *start* flag is lost. Then, the *end* flag as well as the activation zone make their tour around the ring. Note that this is possible for the activation zone since the *start* flag is no longer there to block rule applications. Then the detector turns again passive and creates another *start* flag that reacts with the *end* flag of the previous round, leading to an erroneous *termination* message.

5 Conclusion

We have shown how graph transformation systems with negative application conditions can be viewed as well-structured transition systems. This is similar to the case of lossy vector addition systems [3] where lossiness is used in a similar way in order to deal with inhibiting conditions.

Furthermore we have described a generic backwards search decision procedure and proved that it terminates in specific cases. Termination depends on the existence of bounding functions and a question left open in this paper is to prove their existence in the general case.

Once this problem is solved, it could be interesting to study also general nested application conditions [13,9] and to check whether the results of this paper can be generalized. Furthermore we plan to look at other kinds of well-quasi-orders for graphs, possibly for restricted sets of graphs (see [6]).

We also plan to extend an existing implementation to deal with negative application conditions.

References

1. Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS '96*, pages 313–321. IEEE, 1996.
2. Nathalie Bertrand, Giorgio Delzanno, Barbara König, Arnaud Sangnier, and Jan Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *Proc. of RTA '12*, LIPIcs. Schloss Dagstuhl – Leibniz Center for Informatics, 2012.
3. Ahmed Bouajjani and Richard Mayr. Model checking lossy vector addition systems. In *Proc. of STACS '99*, pages 323–333. Springer, 1999. LNCS 1563.
4. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* Monographs in Theoretical Computer Science. Springer, 2006.
5. Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets. Technical Report RS-94-8, BRICS, May 1994.
6. Michael R. Fellows, Danny Hermelin, and Frances A. Rosamond. Well-quasi-orders in subclasses of bounded treewidth graphs. In *Proc. of IWPEC '09 (Parameterized and Exact Computation)*, pages 149–160. Springer, 2009. LNCS 5917.
7. Alain Finkel and Phillipe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
8. Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Inf.*, 26(3-4):287–313, June 1996.
9. Annegret Habel and Karl-Heinz Pennemann. Nested constraints and application conditions for high-level structures. In *Formal Methods in Software and Systems Modeling. Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday*, pages 294–308. Springer, 2005. LNCS 3393.
10. S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. Technical Report 2012-01, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, 2012.
11. Salil Joshi and Barbara 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.
12. Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.
13. Arend Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.
14. Neil Robertson and Paul Seymour. Graph minors XX. Wagner's conjecture. *Journal of Combinatorial Theory Series B*, 92:325–357, November 2004.
15. Neil Robertson and Paul Seymour. Graph minors XXIII. Nash-Williams' immersion conjecture. *Journal of Combinatorial Theory Series B*, 100:181–205, March 2010.
16. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations.* World Scientific, 1997.

A Proofs

Lemma 6 (Reordering of minor decompositions). *Every minor morphism μ can be decomposed into a finite sequence of one-step minor morphisms μ_i such that $\mu = \mu_1 \circ \ldots \circ \mu_n$ and there are k, ℓ with $1 \leq k \leq \ell \leq n + 1$ and*

1. *μ_i with $1 \leq i < k$ is undefined on one node (node deletion),*
2. *μ_i with $k \leq i < \ell$ is undefined on one edge and not injective on nodes (edge contraction)* and
3. *μ_i with $\ell \leq i < n + 1$ is undefined on one edge and injective on nodes (edge deletion).*

Proof. If μ is the identity, the empty sequence satisfies all conditions. So let $\mu \colon G \mapsto M$ be a minor morphism which is undefined on at least one element. According to Lemma 3 M is a minor of G, hence there is a sequence of node deletions, edge deletions and edge contractions such that some M' isomorphic to M can be obtained from G by application of this sequence. Each element of the sequence corresponds to a one-step minor morphism, thus we obtain a sequence of one-step minor morphisms μ_i such that $\mu = \mu_1 \circ \ldots \circ \mu_n \circ \iota$, where $\iota \colon M' \to M$ is a isomorphism. Note that $\mu_n \circ \iota$ is a one-step minor morphism.

Trivially the deletion of all nodes can be done after all edge deletions and contractions, since a node must be isolated to be deleted (in one step) and can therefore be mapped injectively in every one-step minor morphism contracting or deleting an edge, instead of being deleted directly.

Let $\mu_i \colon X_i \mapsto X_{i+1}$ and $\mu_{i+1} \colon X_{i+1} \mapsto X_{i+2}$ be one-step minor morphisms for some $1 \leq i \leq n - 1$. Let $e_C, e_D \in X_i$ such that μ_i contracts e_C and μ_{i+1} deletes $\mu_i(e_D)$. Then the morphisms $\overline{\mu}_i \colon X_i \mapsto \overline{X}_{i+1}$, $\overline{\mu}_{i+1} \colon \overline{X}_{i+1} \mapsto X_{i+2}$ defined as

$$\overline{\mu}_i(x) = x \text{ if } x \neq e_D \text{ and undefined otherwise}$$

$$\overline{\mu}_{i+1}(x) = \mu_{i+1} \circ \mu_i \circ \overline{\mu}_i^{-1}$$

are one-step minor morphisms with $\mu_{i+1} \circ \mu_i = \overline{\mu}_{i+1} \circ \overline{\mu}_i$ and $\overline{\mu}_i$ deletes an edge, while $\overline{\mu}_{i+1}$ contracts an edge. □

Proposition 1. *A GTS containing rules with negative application conditions is well-structured wrt. the minor ordering if for every rule $r \colon L \rightharpoonup R$ with negative application conditions $n_i \colon L \to N_i$ and every minor morphism $\mu \colon G \mapsto M$ the following holds: for every match $m \colon L \to M$ satisfying all NACs (indicated by the crossed-out arrows), G can be rewritten to some graph G' such that there is a match $m' \colon L \to G'$ satisfying all NACs and a minor morphism $\mu' \colon G' \mapsto M$ with $m = \mu' \circ m'$ (see diagram below).*

$$\begin{tikzcd}
G \arrow[dr, mapsto, bend right, "\mu"'] & L \arrow[r, tail, "n_i"] \arrow[d, tail, "m"] & N_i \arrow[dl, tail, bend left, "/" marking] \\
 & M &
\end{tikzcd}
\qquad
\begin{tikzcd}
G \arrow[dr, Rightarrow, "*"] \arrow[ddr, mapsto, bend right, "\mu"'] & L \arrow[r, tail, "n_i"] \arrow[d, tail, "m'"] \arrow[dd, tail, bend left=50, "m"] & N_i \arrow[dl, tail, bend left, "/" marking] \arrow[ddl, tail, bend left, "/" marking] \\
 & G' \arrow[d, mapsto, "\mu'"] & \\
 & M &
\end{tikzcd}$$

Proof. We have to show that the compatibility condition of Definition 2 ist satisfied. Let M be a graph which is rewritten to H. Then there is a rule r and a match m such that the square in the left diagram below is a pushout. Let $G \geq M$ be a graph, then by Lemma 3 there is $\mu\colon G \mapsto M$.

$$\begin{tikzcd}
N & L \arrow[l, tail, "n_i"'] \arrow[r, "r"] \arrow[d, tail, "m'"] \arrow[dd, bend right, tail, "m"'] & R \arrow[dd] \\
& G' \arrow[d, mapsto, "\mu'"] & \\
& M \arrow[r] & H
\arrow[from=1-1, to=2-2, tail, "/" marking]
\arrow[from=1-1, to=3-2, tail, bend right, "/" marking]
\end{tikzcd}
\qquad
\begin{tikzcd}
N & L \arrow[l, tail, "n_i"'] \arrow[r, "r"] \arrow[d, tail, "m'"] \arrow[dd, bend right, tail, "m"'] & R \arrow[d] \\
& G' \arrow[d, mapsto, "\mu'"] \arrow[r, "r'"] & H' \arrow[d, mapsto, "\mu''"] \\
& M \arrow[r] & H
\arrow[from=1-1, to=2-2, tail, "/" marking]
\arrow[from=1-1, to=3-2, tail, bend right, "/" marking]
\end{tikzcd}$$

By assumption m can be split into a minor morphism $\mu'\colon G' \mapsto M$ and a match $m'\colon L \to G'$ satisfying all NACs, such that $G \Rightarrow^* G'$. As shown in the diagram on the right the pushout (outer square) can be split into two pushouts. Because μ' is a minor morphism and μ' does not contract edges which r' deletes, we can conclude from the variation of Lemma 4 that μ'' is a minor morphism. Note that μ' cannot contract edges r' deletes, because such edges have a preimage in L and therefore an image in M.

Thus, any graph $G \geq M$ can first be rewritten to G' and then to H' such that $H' \geq H$. Hence the compatibility condition is fulfilled. □

Theorem 2. *The procedure pbn*(S) *(Procedure 1) computes a finite subset of Pred*($\uparrow S$).

Proof. Obviously *pbn*(S) is finite. Let $G' \in$ *pbn*(S) be a graph generated by the procedure. Then there is a graph G, a rule $r\colon L \to R$, a match $\overline{m}\colon L \to G'$ and two minor morphisms $\mu\colon R \mapsto M$ and $\nu\colon G' \mapsto G$ (see diagram on the left below). The graph G' is generated by expanding some graph G first to produce an injective match and then to generate a graph where all negative application conditions are satisfied. In both cases decontractions were used, hence ν indicated below is a minor morphism.

$$\begin{tikzcd}
& L \arrow[r, "\mu \circ r"] \arrow[dl, tail, "\overline{m}"'] \arrow[d, "m"] & M \arrow[d] \\
G' \arrow[r, mapsto, "\nu"] & G \arrow[r] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, tail, "\overline{m}"] \arrow[dd, bend right=60, "m"'] & R \arrow[r, mapsto, "\mu"] \arrow[d, tail, "\overline{m}'"'] \arrow[dd, bend left=60, "m''"] & M \arrow[dd, "m'"] \\
G' \arrow[r, "r'"] \arrow[d, mapsto, "\nu"] & H' \arrow[d, mapsto, "\nu'"'] & \\
G \arrow[r] & H'' \arrow[r, mapsto] & S
\end{tikzcd}$$

By definition the match $\overline{m}$ is total and injective, but $m = \nu \circ \overline{m}$ may be non-injective. The application of r rewrites G' so some H' as seen in the right diagram above. We can then form the pushout of r' and ν, to obtain H''. Note that by pushout splitting the right square in the right diagram above is a pushout. Furthermore $m'' = \nu' \circ \overline{m}'$ is total since $\overline{m}'$ is total and ν' only contracts edges not contained in R. Hence (by Lemma 4) S must be a minor of H''. It holds that $H' \geq H'' \geq S$ and hence the application of r to G' yields a graph larger than (or equal to) S. □

Theorem 3. *If there is a bounding function b_r for every rule r, the set generated by $pbn(S)$ is a finite basis of $\uparrow Pred(\uparrow S)$.*

Proof. Let $\hat{G} \in \uparrow Pred(\uparrow S)$, then there is a graph $G \leq \hat{G}$, a rule $r \colon L \rightharpoonup R$ and a total, injective match $m \colon L \to G$ satisfying all NACs, such that G is rewritten to some H and $S \leq H$. By Lemma 3 there are minor morphisms $\nu \colon \hat{G} \mapsto G$ and $\mu' \colon H \mapsto S$. Since a rewriting step is a pushout, by Lemma 5 there are morphisms μ, μ_r, r'' and m'' as shown below, such that the outer square is a pushout.

$$\begin{tikzcd}
N \arrow[dr, tail, "\times" marking] & L \arrow[l, tail, "n_i"'] \arrow[r, harpoon, "r"] \arrow[d, tail, "m"] & R \arrow[d, "m'"] & \\
\hat{G} \arrow[r, mapsto, "\nu"] & G \arrow[r, harpoon, "r'"] & H \arrow[dr, mapsto, "\mu'"] & \\
 & & & S
\end{tikzcd}
\qquad
\begin{tikzcd}
N \arrow[dr, tail, "\times" marking] & L \arrow[l, tail, "n_i"'] \arrow[r, harpoon, "r"] \arrow[d, tail, "m"] & R \arrow[d, "m'"] \arrow[r, mapsto, "\mu_r"] & M \arrow[dd, "m''"] \\
\hat{G} \arrow[r, mapsto, "\nu"] & G \arrow[r, harpoon, "r'"] \arrow[d, mapsto, "\mu"] & H \arrow[dr, mapsto, "\mu'"] & \\
 & X \arrow[rr, harpoon, "r''"] & & S
\end{tikzcd}$$

The morphism $\mu \circ m$ is total and injective on edges, but not necessarily injective on nodes, hence in Step 3 of the procedure non-injective matches are split in every possible way until the match is injective. By Lemma 6, μ can be decomposed in such a way that all edge deletions can be performed before all edge contractions. Thus, while decontracting (and possibly adding previously non-existent nodes) no edge insertions are necessary between decontraction steps. In Step 3 for every X a graph X' is generated such that there are minor morphisms $\mu_2 \colon G \mapsto X'$, $\mu_1 \colon X' \mapsto X$ with $\mu = \mu_1 \circ \mu_2$ and $\mu_2 \circ m$ is a total, injective match of L in X' (see diagram below).

If $\mu_2 \circ m$ satisfies all NACs, X' is stored and the procedure terminates. Since $X' \leq G \leq \hat{G}$, it holds that $\hat{G} \in \uparrow pbn(S)$. If $\mu_2 \circ m$ violates at least one NAC, we observe that any graph larger than X' wrt. the subgraph relation also violates the NAC, since there is a total, injective match of any graph into its larger graph. Thus the match of a NAC can only be destroyed by decontracting an edge. Again by Lemma 6 no edge insertions are needed between decontraction steps.

$$\begin{tikzcd}
\hat{G} \arrow[r, mapsto, "\nu"] & G \arrow[dr, mapsto, "\mu_2"] \arrow[dd, mapsto, "\mu_2'"'] \arrow[ddrr, mapsto, bend left, "\mu"] & & \\
 & & X' \arrow[dr, mapsto, "\mu_1"] & \\
 & M \arrow[ur, mapsto, "\mu_2''"] \arrow[r, mapsto] & M' \arrow[r, mapsto] & X
\end{tikzcd}$$

Because of the bounding function we know that there is a graph M and minor morphisms $\mu_2'\colon G \mapsto M$, $\mu_2''\colon M \mapsto X'$ with $\mu_2 = \mu_2'' \circ \mu_2'$ and $\|\,\mu_2''\,\| \leq b_r(\mu_2 \circ m)$ (see diagram above). Without loss of generality we assume that μ_2'' is minimal, i.e. there are no $\overline{\mu}_2''\colon G \mapsto \overline{M}$, $\overline{\mu}_2'\colon \overline{M} \mapsto X'$ such that $\mu_2 = \overline{\mu}_2'' \circ \overline{\mu}_2'$ and $\|\,\overline{\mu}_2''\,\| < \|\,\mu_2''\,\|$. This means that μ_2'' does not delete edges (without merging some of its nodes) since μ_2'' would still satisfy the conditions of the bounding function if the edge deletion is omitted and therefore μ_2'' would not be minimal. Since μ_2'' contracts at most $\|\,\mu_2''\,\| \leq b_r(\mu_2 \circ m)$ edges, at some point while recursively processing step 4 a graph M' is reached which is a subgraph of M possibly missing some isolated nodes. Since $M' \leq M \leq G \leq \hat{G}$ it holds that $\hat{G} \in {\uparrow}pbn(S)$.

Since all steps of the procedure have finite runtime, especially the splitting in step 3 is bounded by the number of non-injectively matched nodes and the recursion in step 4 is performed to a fixed depth, the number of stored graphs is finite and hence a finite basis. □

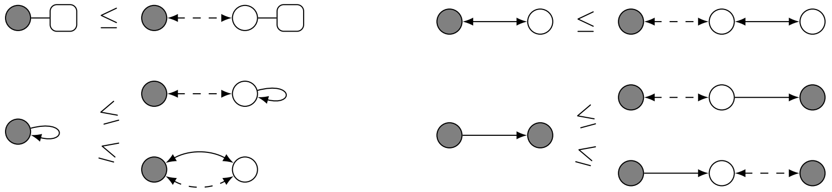
Proposition 2. *There is a bounding function for all rules if the corresponding GTS satisfies the following properties:*

1. *$ar(\ell) \leq 2$ for all labels $\ell \in \Lambda$* and
2. *for every negative application condition $n\colon L \to N$ the graph N has at most one edge and two nodes in addition to $n(L)$.*

Proof. Let $n_i\colon L \to N$ be a NAC. If every edge of N has a preimage in L or if the edge in N which has no preimage in L is only connected to nodes which have no preimage in L, n_i cannot be destroyed by decontractions (or in any other way).

For each rule r we define the result of the bounding function $b_r(m)$ for some match $m\colon L \to G$ to be zero if there is at least one such NAC and the sum of the number of different matches of each NAC in G if r has no such NAC.

If a rule has at least one such NAC, the conditions for a bounding function are trivially satisfied (no larger graph exists satisfying all NACs). So assume every NAC is of one of the following forms: a unary edge connected to a node with preimage, a binary edge connected the one node with preimage, a binary edge connected to two different nodes with preimage or a binary edge connected to one node with preimage and one node without preimage. All possibilities to destroy a match of one such NAC using decontraction is shown below. Gray nodes indicate nodes with preimage in L, dashed edges are the edges added by the decontraction and arrows with two arrow tips indicates that the arrow can be directed in both ways.



Each match of a NAC can be destroyed with just one decontraction, but this decontraction may generate a new match for the same or a different NAC. Let G' be a graph and $\mu\colon G' \mapsto G$ a minor morphism such that there is a match $m'\colon L \to G'$ satisfying all NACs and $m = \mu \circ m'$. We assume that G' is minimal, that is there is no graph G'' with minor morphisms $\mu'\colon G' \mapsto G''$, $\mu''\colon G'' \mapsto G$ such that $\mu' \circ m'$ is a match satisfying all NACs. Each edge $e \in G'$ on which μ is undefined is attached to at least one node in the range of m', since otherwise e can be deleted or contracted without generating a match of any NAC, because for that at least one node has to be in the range of m'. Now if a new match of a NAC is generated by a decontraction the added edge e' is part of it and the match has to be deleted again by decontraction as shown above. In every case μ will be undefined for e' and e' is attached to nodes not in the range of m'. The other decontraction could also be performed first, thus every decontraction generating a new match of a NAC is already represented by another case and can be discarded. □