Termination Analysis for Graph Transformation Systems

H. Bruggink, Barbara König, Hans Zantema

Termination Analysis for Graph Transformation Systems

H.J. Sander Bruggink[1], Barbara König[1], and Hans Zantema[2]

[1] Universität Duisburg-Essen
{sander.bruggink,barbara_koenig}@uni-due.de
[2] Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen
h.zantema@tue.nl

Abstract. We introduce two techniques for proving termination of graph transformation systems. We do not fix a single initial graph, but consider arbitrary initial graphs (uniform termination), but also certain sets of initial graphs (non-uniform termination). The first technique, which can also be used to show non-uniform termination, uses a weighted type graph to assign weights to graphs. The second technique reduces uniform termination of graph transformation systems of a specific form to uniform termination of cycle rewriting, a variant of string rewriting.

1 Introduction

Termination, the absence of infinite computations, is a property that is required in many applications, in particular in model transformation, algorithms and protocol specifications. Many of these applications, such as graphical model transformation (for example, of UML models) and algorithms that manipulate data structures on the heap, are naturally modeled by graph transformation systems. This paper is concerned with the termination of such graph transformation systems. In particular we study the following question: given a set of graph transformation rules, and possibly an infinite set of potential initial graphs, does a transformation sequence of infinite length from one of the initial graphs exist? This problem is undecidable in general [9], but it is still important to develop semi-decision procedures that correctly decide as many instances as possible (and output "unknown" in the others).

Although termination is a basic notion of any computational formalism, it has not received a lot of attention in the graph transformation community; the focus is on reachability problems – the question whether a graph with some required or unwanted property is reachable from an initial graph. However, some prior work on the topic exists, mostly applied to model transformation [2, 6, 10]. Similar to [3] we follow a more general approach. We consider graph transformation from a theoretical point of view. This has the disadvantage of making results harder to obtain, but the advantage of being more broadly applicable.

The paper is organized as follows. In Sect. 2 we recapitulate definitions and fix notation. The heart of the paper is formed by Sect. 3, in which we introduce the

weighted type graph technique for proving termination of graph transformation systems. We define the technique, consider special cases and investigate its limits; finally we give a detailed example that demonstrates its strengths. In Sect. 4 we show that termination of graph transformation systems of a specific kind can be reduced to termination of cycle rewriting, which is a form of rewriting that is related to string rewriting. This clarifies the relation to string rewriting and provides us with an additional method for graphs. Finally, we briefly present an implementation of the techniques of this paper in Sect. 5, compare with related work in Sect. 6 and give pointers for further research in Sect. 7. A full version of this paper with proofs and additional details can be downloaded from the first author's website.

2 Preliminaries

We first introduce graphs, morphisms, and graph transformation, in particular the double pushout approach [5]. We use edge-labeled, directed graphs.

Definition 1 (Graph). *Let Λ be a fixed set of edge labels. A* Λ-labeled graph *is a tuple $G = \langle V, E, src, tgt, lab \rangle$, where V is a set of nodes, E is a set of edges, $src, tgt \colon E \to V$ assign to each edge a source and a target, and $lab \colon E \to \Lambda$ is a labeling function.*

As a notational convention, we will denote, for a given graph G, its components by V_G, E_G, src_G, tgt_G and lab_G, unless otherwise indicated. The size of a graph G, written $|G|$, is the number of nodes and edges it contains, that is $|G| = |V_G| + |E_G|$.

Definition 2 (Graph morphism). *Let G, G' be two Λ-labeled graphs. A* graph morphism *$\varphi \colon G \to G'$ consists of two functions $\varphi_V \colon V_G \to V_{G'}$ and $\varphi_E \colon E_G \to E_{G'}$, such that for each edge $e \in E_G$ it holds that $src_{G'}(\varphi_E(e)) = \varphi_V(src_G(e))$, $tgt_{G'}(\varphi_E(e)) = \varphi_V(tgt_G(e))$ and $lab_{G'}(\varphi_E(e)) = lab_G(e)$.*

We will often drop the subscripts V, E and simply write φ instead of φ_V, φ_E.

Definition 3 (Graph transformation). *A* graph transformation rule *ρ consists of two injective morphisms $L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$, consisting of the* left-hand side *L, the* right-hand side *R and the* interface *I.*

A match *of a left-hand side in a graph G is an injective morphism $m \colon L \to G$.*

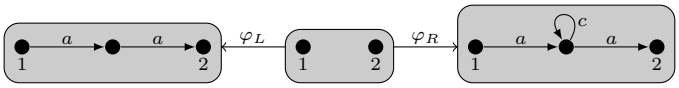
Given a rule ρ and a match $m \colon L \to G$, a graph H is the result of applying the rule at the match, written $G \Rightarrow_{m,\rho} H$ (or $G \Rightarrow_\rho H$ if m is arbitrary or clear from the context), if there exists a graph C and injective morphisms such that the two squares in the diagram on the right are pushouts in the category of graphs and graph morphisms.

$$\begin{array}{ccccc} L & \xleftarrow{\varphi_L} & I & \xrightarrow{\varphi_R} & R \\ {\scriptstyle m}\downarrow & (\mathrm{PO}) & \downarrow & (\mathrm{PO}) & \downarrow \\ G & \longleftarrow & C & \longrightarrow & H \end{array}$$

A graph transformation system *$\mathcal{R}$ is a set of graph transformation rules. For a graph transformation system $\mathcal{R}$, $\Rightarrow_\mathcal{R}$ is the rewriting relation on graphs induced by those rules.*

The above formal definition of a graph transformation step can be algorithmically described as follows. Let a rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ and a match $m\colon L \to G$ be given. The rule can be applied to the match if for every edge e of G which is not in the image of m it holds, for $v \in \{src(e), tgt(e)\}$, that v has a pre-image in I (under $m \circ \varphi_L$) if v has a pre-image in L (under m). In this case we say that the *dangling edge condition* is satisfied. If the rule is applicable, all images of elements in L, whose preimages are not in the interface I, are removed from G. This gives us the "context" graph C. Furthermore the elements of R that do not have a preimage in I are added and connected with the remaining elements, as specified by the interface. This results in the graph H. The dangling edge condition ensures that nodes can only be deleted if all incident edges are deleted.

Example 1. We take as label set $\Lambda = \{a, c\}$. Consider the following graph transformation rule:



The numbers represent which nodes are mapped to each other. The following is a legal transformation step using the above rule:

There is no step replacing the *aa*-pattern at the bottom, because the middle node, although deleted by the rule, is incident to a c-edge not in the pattern (the dangling edge condition is not satisfied, that is, edges would be left dangling).

A graph is discrete when it does not contain any edges. A well-known result from double-pushout graph transformation is that we can restrict to rules with discrete interfaces without affecting the expressive power of the formalism: for each rule with non-discrete interface a rule with discrete interface exists which induces the same rewrite relation. As examples we will only use rules with discrete interfaces, although the results of Sect. 3 are also applicable to graph transformation systems that contain rules with non-discrete interfaces.

Let $\mathcal{L}$ be a set of graphs. We say that a set of rules $\mathcal{R}$ is $\mathcal{L}$-*terminating* if each reduction sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} \cdots$ with $G_0 \in \mathcal{L}$ is finite. The set of rules $\mathcal{R}$ is *uniformly terminating* or simply *terminating* if it is $\mathcal{G}$-terminating, where $\mathcal{G}$ is the set of all graphs.

We will specify sets of graphs, in this setting called *graph languages*, by so-called type graphs.[3] A type graph is just a graph T. The graph language accepted by T, written $\mathrm{L}(T)$, is the set of all graphs from which there exists a morphism into

[3] In the literature, for example [4], typing morphisms are often considered as an intrinsic part of graph and rule definitions. We consider untyped graphs and rules, and use type graphs merely as a means to describe graph languages.

T, that is: $\mathrm{L}(T) = \{G \mid \text{there exists a morphism } \varphi\colon G \to T\}$. A type graph T is closed under a set of rules $\mathcal{R}$, if for each rule $L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R \in \mathcal{R}$ and morphism $t_L\colon L \to T$, there exists a morphism $t_R\colon R \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$. A type graph T being closed under a set of rules $\mathcal{R}$ ensures that $\mathrm{L}(T)$ is closed under $\mathcal{R}$-reachability, that is, it ensures that if $G \Rightarrow_{\mathcal{R}} H$ and $G \in \mathrm{L}(T)$, then also $H \in \mathrm{L}(T)$.

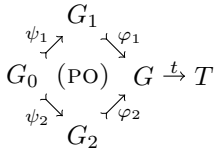
3 Termination Analysis via Weighted Type Graphs

In this section, we present a termination argument based on weighted type graphs. The technique is inspired by the semantic labeling technique for proving termination of term rewrite systems [11], where the algebra is replaced by type graphs.

3.1 Weighted Type Graphs

We assume a set W of *weights* with a binary operation $\oplus$ and a well-founded partial order $<$ such that the following holds: for $a, b, c \in W$ we have that (i) $a < b \iff a \oplus c < b \oplus c$, and (ii) $a = b \iff a \oplus c = b \oplus c$. Note that from these two conditions it follows that (iii) $a \leq b \iff a \oplus c \leq b \oplus c$.

Furthermore, for a given graph T, called the *type graph*, we have a weight function w that assigns a weight from W to every morphism $\varphi\colon G \to T$. This weight function must be stable under pushouts in the following sense: given a pushout of injective morphisms as shown on the right and an arrow $t\colon G \to T$ from the pushout object G into the type graph, we have that $w(t) \oplus w(t \circ \varphi_1 \circ \psi_1) = w(t \circ \varphi_1) \oplus w(t \circ \varphi_2)$. (Note that $\varphi_1 \circ \psi_1 = \varphi_2 \circ \psi_2$.)



The intuiton behind stability under pushouts is the following. A pushout can be seen as an operation which glues together two graphs over a common interface. Above, the graphs G_1 and G_2 are glued together over the common interface G_0. The weight of morphisms from the G to T should be obtained by adding together the weights of the corresponding morphisms from G_1 and G_2 to T. However, in $w(t \circ \varphi_1) \oplus w(t \circ \varphi_2)$ the common interface is counted twice, so we have to add $w(t \circ \varphi_1 \circ \psi_1)$ to the left-hand side to balance the equation.

Although the termination argument will be stated for weight functions of this form in general, the specific type of weight function we will use in examples and in Sect. 3.4 will be so-called *linear weight functions*, which are defined as follows. Weights are natural numbers with summation and order $<$. Let $d\colon (V_T \cup E_T) \to \mathbb{N}$ be a function which assigns a weight to each node and edge of the type graph T. The linear weight function w_d for d assigns to a morphism $\varphi\colon G \to T$ the weight $w_d(\varphi) = \sum_{x \in (V_G \cup E_G)} d(\varphi(x))$.

Proposition 1. *Let T be a type graph and $d\colon (V_T \cup E_T) \to \mathbb{N}$ a function assigning a weight to all nodes and edges of T. The linear weight function w_d as defined above is a well-defined weight function, that is, it is stable under pushouts.*

Now termination analysis works as follows.

Definition 4. *Let T be a type graph and w a weight function for T.*

(i) A rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ is tropically decreasing *with respect to T and w if for each morphism $t_L \colon L \to T$ (where t_L is not necessarily injective), there exists a morphism $t_R \colon R \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$ and $w(t_L) > w(t_R)$. A rule ρ is* tropically non-increasing *if the same condition applies, except that $w(t_L) \geq w(t_R)$.*

(ii) A rule $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ is arctically decreasing *with respect to T and w if for each morphism $t_R \colon R \to T$ (where t_R is not necessarily injective), there exists a morphism $t_L \colon L \to T$ such that $(t_L \circ \varphi_L) = (t_R \circ \varphi_R)$ and $w(t_L) > w(t_R)$. A rule ρ is* arctically non-increasing *if the same condition applies, except that $w(t_L) \geq w(t_R)$.*

Note that in the definition above the morphisms into type graphs are not necessarily injective, although the morphisms used in rules and matches are. This is intended, because only for subgraphs T' of the typegraph T (or graphs isomorphic to such a subgraph) there exists an injective morphism from T' to T, and restricting to the subgraphs of T is clearly undesired.

The names *tropical* and *arctic* stem from string rewriting, where analogous termination arguments use tropical and arctic semi-rings as evaluation algebras; see for example [8].

Theorem 1. *Let $\mathcal{L}$ be a set of graphs and let $\mathcal{R}$ be a graph transformation system. Furthermore, let T be a type graph which is closed under $\mathcal{R}$, such that $\mathcal{L} \subseteq \mathrm{L}(T)$, and let w be a weight function of T.*

Finally, let $\mathcal{R}' \subseteq \mathcal{R}$ be such that one of the following conditions holds:

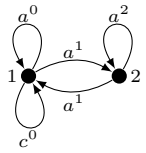
- *all rules of $\mathcal{R}'$ are tropically decreasing with respect to T and w, and all rules of $\mathcal{R} \setminus \mathcal{R}'$ are tropically non-increasing with respect to T and w; or*
- *all rules of $\mathcal{R}'$ are arctically decreasing with respect to T and w, and all rules of $\mathcal{R} \setminus \mathcal{R}'$ are arctically non-increasing with respect to T and w.*

Then $\mathcal{R}$ is $\mathcal{L}$-terminating if and only if $\mathcal{R} \setminus \mathcal{R}'$ is $\mathcal{L}$-terminating.

The above theorem allows to "remove" rules from a graph transformation system, concluding termination of the complete system from termination of a subset of the rules (this is called *relative termination* in the literature). In the case that $\mathcal{R} = \mathcal{R}'$ termination follows directly. Otherwise, a new termination argument for the simpler system is sought. Thus, we obtain iterative termination proofs (see Sect. 3.5 for an example of such an iterative proof).

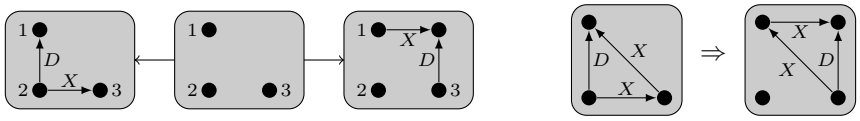
Example 2. See the graph transformation system from Ex. 1. This system is terminating because of the dangling edge condition, because the number of nodes without a c-loop strictly decreases in each transformation step. Now, we want to use the type graph argument to prove this. Since there is only one rule, it is sufficient to show that it is decreasing with respect to some type graph T and some weight function.

We take the type graph and weight function d displayed on the right. The superscripts of the edge labels denote the d-value of the edge, while $d(v) = 0$ for all nodes v. Because of the "flower structure" on the left node, every graph consisting of a- and c-labeled edges can be mapped into this type graph, so we can show uniform termination.



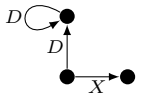
We arctically evaluate the rule with respect to the given type graph. There are a number of morphisms from the right-hand side into the type graph. For each of them a corresponding morphism of greater weight from the left-hand side into the type graph, which agrees on the interface nodes, can be found. For example, one possibility is to map nodes 1 and 2 of the right-hand side of the rule to node 2 of the type graph and the middle node of the right-hand side to node 1 of the type graph; the edges are mapped accordingly (weight: 2). A compatible morphism from the left-hand side to the type graph maps all nodes to node 2 of the type graph (weight: 4). Thus, the system terminates by Theorem 1.

Example 3. Let $\Lambda = \{D, X\}$. Consider the graph transformation system which consists of the rule depicted below on the left. This graph transformation system is not uniformly terminating, as is witnessed by the step displayed on the right, the target of which contains its source as a subgraph:

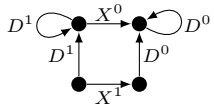


It is $\mathrm{L}(S)$-terminating, however, where S is given on the right. This follows by considering the weighted type graph T, where the weights of the edges are again given by the superscripts. For each morphism from the left-hand side into T, a smaller morphism from the right-hand side into T, which agrees on the interface nodes, can be found, so the rule is tropically decreasing. Since T contains S as a subgraph and is closed under the transformation rule, the transformation system is $\mathrm{L}(S)$-terminating by Theorem 1.

$S =$



$T =$



3.2 Special Case: Node and Edge Counting

A simple (but weak) termination argument for graph transformation (previously considered in, among others, [1, 2]) is the counting of nodes and edges. We consider a somewhat more general variant, *weighted* node and edge counting.

Let $d\colon \Lambda \to \mathbb{N}$ be a function which maps each label to a weight. Then a graph G can be assigned a weight by taking the sum of the weights of all labels occurrences in it: $w(G) = \sum_{e \in E_G} d(\ell_G(e))$. If for each production $L \leftarrow I \rightarrow R$ it holds that $w(L) > w(R)$, then the graph transformation system terminates.

This termination argument is a special case of the weighted type graph argument. As type graph we take the "flower" graph $F_\Lambda = \langle \{v\}, \Lambda, s, t, \ell \rangle$ with

for each $A \in \Lambda$: $s(A) = v$, $t(A) = v$ and $\ell(A) = A$. Now we can use the linear weight function w_d, as defined before Prop. 1. This works with both the tropical and the arctic variant of the weighted type graph termination argument.

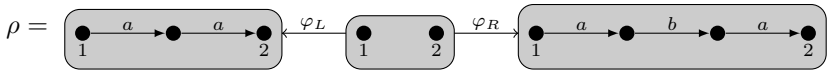
With an even simpler argument in the same style we can view node counting as a special case of the weighted type graph approach.

3.3 Special Case: Match-Bounds

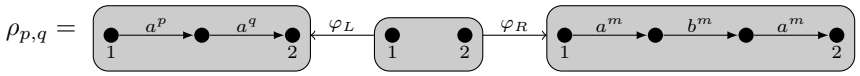
In [3] a method for proving termination of graph transformation systems based on the match-bounds approach for string rewrite systems [7] was introduced. Here, we briefly recapitulate the termination argument of this paper and show that it can be considered as a special case of the weighted type graph approach.

The idea of match-bounds is to annotate the edges with a "creation height", which is a measure of how many previous transformation steps were responsible for creating the edge. In particular, when an occurrence of a left-hand side is replaced by a right-hand side, the new edges are annotated with a creation height which is equal to the smallest creation height of the left-hand side plus one. Now, the termination argument is as follows: if there exists a type graph (with annotated edges) which is closed under the annotated graph transformation system, then the original graph transformation system is terminating. See [3] for more details and formal definitions and proofs.

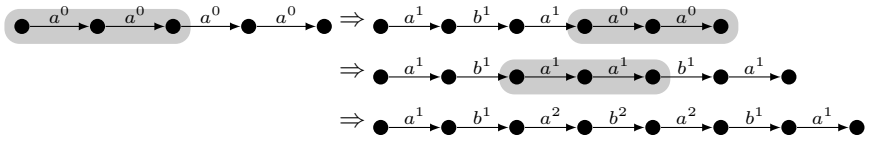
Example 4. Consider the graph transformation consisting of the following rule:



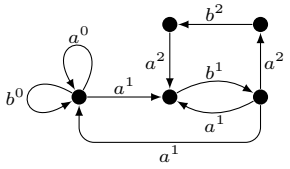
This rule is replaced by the infinitely many annotated rules of the following form, where $m = \min(p, q) + 1$. The superscripts denote the creation height annotations of the edges.



The following is an annotated transformation sequence of this system:



The following is a type graph which is closed under the annotated system:



So, by the match-bounds technique, we conclude that the system is terminating.

The match-bounds approach is a special case of the type graph approach, in particular of the tropical variant. The annotated type graph plays the role of the type graph by which graphs are assigned weights. Consider an annotated type graph T with maximum annotation c (in Ex. 4 above, $c = 2$). Let T' be the graph which is equal to T, except that the annotations are removed from the labels. As weights we take strings of natural numbers of length $c + 1$, lexicographically ordered. A morphism $\varphi\colon G \to T'$ is now assigned a weight as follows: $w(\varphi) = n_0 \dots n_c$, where n_i is the number of edges in G which are mapped to an edge of T which has annotation i. Analogously to Prop. 1 we can show that this weight function is stable under pushouts and thus well-defined. By construction, all rules are decreasing with respect to this type graph.

3.4 Derivational Complexity

In this subsection we consider the type graph method with linear weight functions.

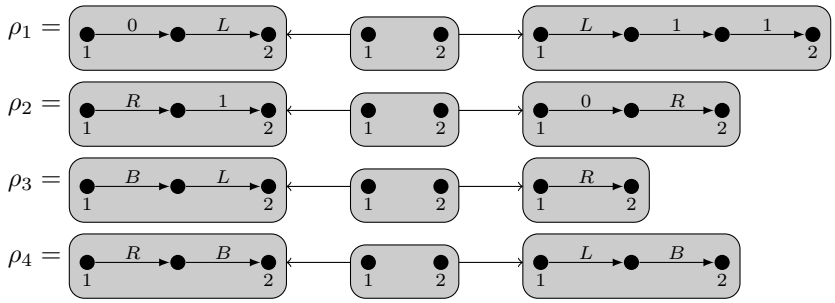
First, we show that transformation sequences of graph transformation systems which can be proved to be terminating with a single application of the type graph technique are linearly bounded (with respect to the size of the initial graph). Because we restrict to linear weight functions, this result is not very surprising.

Secondly, however, we show by example that graph transformation systems that can be proven terminating by a *repeated* application of the type graph technique may even have transformation sequences of exponential length.

Proposition 2. *Let T be a type graph and $d\colon (V_T \cup E_T) \to \mathbb{N}$ a weight function. Furthermore, let $\mathcal{R}$ be graph transformation system such that all rules $r \in \mathcal{R}$ are decreasing with respect to T and w. Then there exists a $c \in \mathbb{N}$ such that for each $\mathcal{R}$-reduction sequence $\boldsymbol{G} = G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \cdots$ it holds that $|\boldsymbol{G}| \leq c \cdot |G_0|$.*

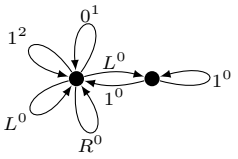
Although proving termination by a single type graph implies a linear reduction bound, by repeating the type graph argument we can show termination of systems with even exponential reduction bounds, as the following example shows:

Example 5. Consider the following graph transformation system, adapted from cycle rewriting [12], which consists of the following graph transformation rules:



This graph transformation system has exponential derivational complexity: starting from a string graph of the form $B^n R 1 B$ the string graph of the form $L 1^{2^n} B$ is reachable in exponentially many steps.

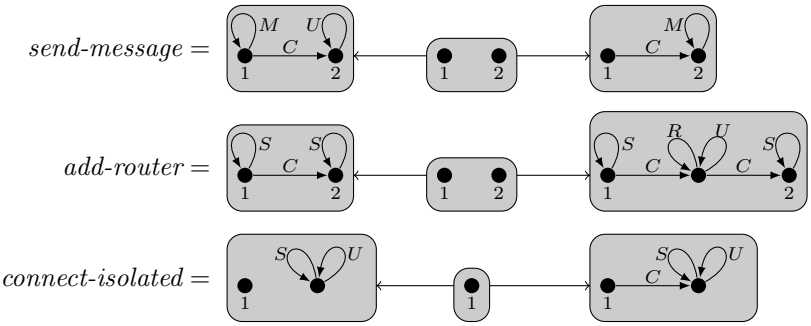
We can show that this graph transformation system terminates. First, ρ_3 can be removed by counting B's, then ρ_4 by counting R's – as shown in Sect. 3.2, label counting is an instance of the type graph technique. For ρ_1 and ρ_2, which indeed by itself have a linear derivational complexity, we construct the type graph on the right (using tropical evaluation).



Note that, although the example derives from a similar example in cycle rewriting, the termination proof is stronger, since it shows uniform termination for all possible start graphs.

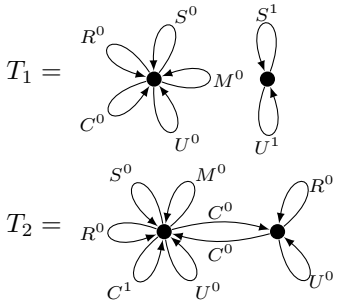
3.5 Detailed Example: Ad-hoc Routing Protocol

We conclude the paper by demonstrating the weighted type graph technique on a simple ad-hoc routing protocol in a dynamically changing network. A message (M) traverses a network of servers (S), routers (R) and directed connections (C) between them. The message can only be sent to unvisited (U) nodes. In addition, rules which modify the network's layout are active. The graph transformation system which models the protocol consists of the following rules:



Note that, due to the dangling edge condition, the *connect-isolated* rule cannot be applied to patterns where the right node is connected to any other nodes; in fact this condition ensures termination in this case.

The fact that this system is uniformly terminating can be shown using a sequence of weighted type graph arguments. First, we arctically evaluate the rules with respect to T_1 (again, the superscripts denote the weights of the edges). For *connect-isolated*, the right-hand side can only be mapped to the flower structure on the left node (weight 0), while for the left-hand side the nodes with the S- and the U-loop can be matched to the right node (weight 2). For



the other rules, both the left-hand side and the right-hand side can only be mapped to the flower, so all morphisms have weight 0. Thus, *connect-isolated* can be removed.

Now, we tropically evaluate *send-message* and *add-router* with respect to T_2. In both cases, the left-hand side can only be mapped to the flower structure on the left node (weight: 1). But for *add-router*, a compatible morphism from the right-hand side to T_2 can be found which maps the middle node to the right node of T_2 (weight: 0). Thus, *add-router* can be removed.

Finally, the system consisting of only *send-message* is terminating because the number of U-edges strictly decreases in each step.

4 Termination Analysis via Cycle Rewriting

In this section we consider graph transformation systems of a specific string-like form, and show that termination of such systems reduces to termination of cycle rewriting [12], which is a variant of string rewriting.

The result of this section has both theoretical and practical relevance. From a theoretical point of view, it shows that graph transformation shares some characteristics (with respect to termination) with a string-based rewriting formalism, which motivates considering cycle rewriting as a step between graph transformation and term rewriting. In fact, [12] uses an approach similar to the type graph method for proving termination of cycle rewrite systems. For cycle rewrite systems finding termination arguments is easier because of the more restricted format.

From a practical point of view it is useful for proving termination of actual graph transformation systems. Although graph transformation systems which consist only of string-like rules are rare "in the wild", such rules do occur quite often. We can try to use the weighted type graph method to first remove the non-string-like rules from the system, and then, when only string-like rules are left, apply the easier (faster) techniques for cycle rewriting to finish the termination proof.

4.1 Cycle Rewriting with Graph Transformation Systems

Cycle rewriting, introduced in [12], is a variant of string rewriting where strings are considered modulo cyclic shift. Let Σ be an alphabet (that is, a finite set of symbols). Fur $u, v \in \Sigma^*$, we write $u \sim v$ if there are $u_1, u_2 \in \Sigma^*$ such that $u = u_1 u_2$ and $v = u_2 u_1$. A *cycle rewrite rule* is a pair $\langle \ell, r \rangle$ of strings, written $\ell \to r$, and a *cycle rewrite system* is a finite set $\mathcal{R}$ of cycle rewrite rules. A string s rewrites to a string t (written $s \Rightarrow_{\mathcal{R}} t$) if there are a rule $\ell \to r \in \mathcal{R}$ and $s', t', x, y \in \Sigma^*$ such that $s \sim s'$, $t \sim t'$, $s' = x\ell y$ and $t' = xry$.

First, we have to encode cycle rewrite systems as graph transformation systems. The natural way to do this is to represent a string by a "string graph", a graph consisting of a single path, and a cycle by a "cycle graph", a graph consisting of a single cycle.

Let $w = a_1 \cdots a_n$ (where $a_i \in \Sigma$ for $1 \leq i \leq n$) be a string. We define $\mathit{path}(w) = \langle V, E, \mathit{src}, \mathit{tgt}, \mathit{lab} \rangle$, where $V = \{v_0, \ldots, v_n\}$, $E = \{e_1, \ldots, e_n\}$, $\mathit{src}(e_k) = v_{k-1}$, $\mathit{tgt}(e_k) = v_k$ and $\mathit{lab}(e_k) = a_k$ for $1 \leq k \leq n$.

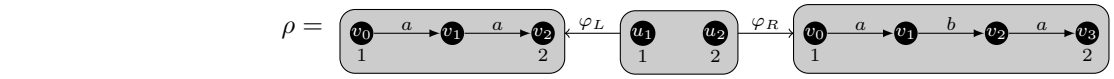
Furthermore, we define $\mathit{cycle}(w) = \langle V, E, \mathit{src}, \mathit{tgt}, \mathit{lab} \rangle$, where $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_n\}$, $\mathit{src}(e_1) = v_n$, $\mathit{tgt}(e_1) = v_1$, $\mathit{lab}(e_1) = a_1$, and $\mathit{src}(e_k) = v_{k-1}$, $\mathit{tgt}(e_k) = v_k$ and $\mathit{lab}(e_k) = a_k$ for $2 \leq k \leq n$.

To encode a cycle rewrite rule, it is natural to encode the left-hand side and the right-hand side with a string graph, and associate via the interface the left-most node and right-most node of the left-hand side with left-most node and right-most node of the right-hand side, respectively. Still, there are two natural choices of what to do with the middle nodes of the left-hand side. Either they are deleted (have no corresponding node in the right-hand side) or they are kept (for each middle node we add an isolated node to the right-hand side). First, we show that under the first encoding termination is preserved. Then, we extend the result to the second encoding by showing that in this case the isolated nodes can be removed from the right-hand side without affecting termination.

Let $\rho = \ell \to r$ be a cycle rewrite rule. We define $\mathit{graph}(\rho) = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$, where $L = \mathit{path}(\ell)$ and $R = \mathit{path}(r)$; $I = \langle \{u_1, u_2\}, \varnothing, \varnothing, \varnothing, \varnothing \rangle$; $\varphi_L(u_1) = v_0$ and $\varphi_L(u_2) = v_{|\ell|}$; and $\varphi_R(u_1) = v_0$ and $\varphi_R(u_2) = v_{|r|}$.

For a cycle rewrite system $\mathcal{R}$, $\mathit{graph}(\mathcal{R})$ consists of the graph transformation rules corresponding to its rules: $\mathit{graph}(\mathcal{R}) = \{\mathit{graph}(\rho) \mid \rho \in \mathcal{R}\}$.

Example 6. Let the rule $aa \to aba$ be given. The corresponding graph transformation rule is:



Above, the white labels inside the nodes are the names given to the nodes in the definitions above, while the numbers below represent the morphisms φ_L and φ_R.

Termination of graph transformation systems of this specific form can now be reduced to termination of cycle rewriting – which, because of the more restricted form, is in some cases slightly easier. Techniques for proving termination of cycle rewrite systems were developed in [12]. Note that here we consider *uniform termination*: the rules in $\mathit{graph}(\mathcal{R})$ are applied to arbitrary graphs, not only to cycles. Hence, proving such a result is non-trivial since we have to derive termination on all graphs from the fact that the rules terminate on all possible cycles.

Theorem 2. *Let $\mathcal{R}$ be a cycle rewrite system. $\mathcal{R}$ is terminating if and only if $\mathit{graph}(\mathcal{R})$ is terminating.*

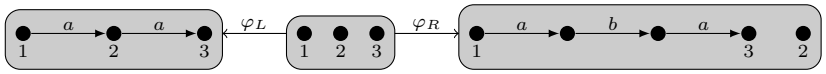
Since termination of cycle rewriting is undecidable (proved in [12]; basically it is a consequence of the undecidability of termination of string rewriting), we obtain an alternative proof of the following result (previously proved in [9]) as a small bonus:

Corollary 1. *Uniform termination of graph transformation systems is undecidable.*

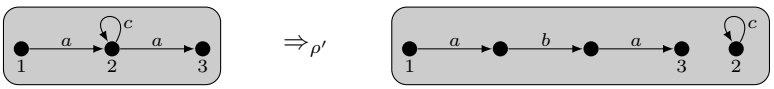
4.2 Removing Isolated Nodes from the Right-Hand Side

Above, we mentioned two different encodings for string rewrite rules: either the middle nodes are deleted or they are maintained by adding corresponding isolated nodes to the left-hand side. (In fact, we can make this choice independently for each middle node.) Which of the encodings we adopt is significant, as the following example shows.

Example 7. Consider the rule ρ of Ex. 6 and the following rule ρ':

$\rho' =$ 

The rules ρ and ρ', although similar, generate a different transformation relation, even if we ignore isolated nodes. The following transformation step is possible with ρ':



Because of the dangling edge condition, there are no ρ-steps at all from the source of the above step.

It turns out that, for a class of graph transformation systems which includes "string-like" systems, isolated nodes can be removed from the right-hand sides without affecting termination of the system.

Let $\rho = L \leftarrow_{\varphi_L} I \rightarrow_{\varphi_R} R$ be a graph transformation rule with a discrete interface I. The graph transformation rule $\mathit{deiso}(\rho)$ is obtained by removing from R all isolated nodes, removing from I all nodes which are mapped by φ_R to an isolated node, and restricting φ_L and φ_R to the new smaller I. For a graph transformation system $\mathcal{R}$, $\mathit{deiso}(\mathcal{R}) = \{\mathit{deiso}(\rho) \mid \rho \in \mathcal{R}\}$.

Proposition 3. *Let $\mathcal{R}$ be a graph transformation system and $\mathcal{C}$ a cycle rewrite system. If $\mathit{deiso}(\mathcal{R}) = \mathit{graph}(\mathcal{C})$, then $\mathcal{R}$ is terminating if and only if $\mathcal{C}$ is terminating.*

Example 8. Consider the graph transformation system $\mathcal{R}$ consisting of the rule ρ' of Ex. 7. The graph transformation system $\mathit{deiso}(\mathcal{R})$ is (isomorphic to) the graph transformation system of Ex. 6 (consisting of ρ), which is $\mathit{graph}(aa \to aba)$. Since $\{aa \to aba\}$ is a terminating cycle rewrite system (see [12]), $\mathcal{R}$ is terminating by Prop. 3.

5 Implementation

A prototype Java-based tool, called *Grez*, has been written, which implements, among others, the termination techniques presented in this paper. The tool may be downloaded from the following web page: `www.ti.inf.uni-due.de/research/tools/grez`.

The tool concurrently runs a number of algorithms. Each of the algorithms tries to prove uniform termination of a graph transformation system. As soon as one of the algorithms successfully finds a termination proof, all algorithms are interrupted and the proof is reported to the user. If the found termination proof is relative, that is, termination of a smaller system must still be proved, this procedure is repeated.

The key algorithms implemented in the tool are:

- *Weighted type graphs.* For all weighted type graphs with a user-specified number of nodes and a user-specified maximum weight it is checked whether they prove the graph transformation system terminating.
- *Cycle rewriting.* If the graph transformation system is of the correct form, it is translated into a cycle rewrite system. Then, the tool TORPAcyc, developed in the context of [12], is run as an external program to prove termination.
- *Node counting.* It is checked whether all left-hand sides have more nodes than the corresponding right-hand side.
- *Label counting.* For all one- and two-element subsets of the labels, and for the set of all labels, it is checked that all left-hand sides have more edges labeled with such a label than the corresponding right-hand side.
- *Match bounds.* The algorithm of [3] is implemented. Additionally, nodes are optionally merged according to two rules: *target-merging*: if a node of the type graph has two outgoing edges with the same label, the target nodes of the edges are merged; and *source-merging*: if a node of the type graph has two incoming edges with the same label, the source nodes of the edges are merged.

Note that the last three techniques are subsumed by the weighted type graph technique. However, specialized algorithms make them often substantially faster than the type graph technique.

We ran the tool on the (uniformly terminating) examples of this paper, using a Linux workstation with a 2.67 Mhz, 4-core CPU. To be better able to compare run times, we only enabled the *weighted type graphs* algorithm (using linear weight functions and both tropical and arctic evaluation). Note that with a single running algorithm the tool is essentially single-threaded. The parameters we used were: generate weighted type graphs with at most two nodes and a maximum weight of 1 (that is: 0 or 1). The run times are listed in Table 1. Note that the ad-hoc routing protocol takes significantly longer than the other examples; this is due to the larger number of labels and thus larger number of type graphs that the exhaustive algorithm needs to generate. It is a direction for further research to develop better heuristical algorithms to find suitable weighted type graphs.

6 Related Work

As mentioned in the introduction, various other works concern themselves with termination of graph transformation, more specifically, of graph transformation as a model transformation formalism.

Example	Run time (s)
Adding c-loop (Ex. 2)	0.20
$graph(aa \to aba)$ (Ex. 4 and Ex. 6)	0.15
Exponential transformation complexity (Ex. 5)	0.93
Ad-hoc routing protocol (Sec. 3.5)	144.8

Table 1. Run times (in seconds) of running the weighted type graph finder of *Grez* on various examples of the paper (average of 5 tries).

The paper [2] considers high-level replacement units (HLRU), which are transformation systems with external control expressions. The paper introduces a general framework for proving termination of such HLRUs, but the only concrete termination criteria considered are node and edge counting, which are subsumed by the weighted type graph method of this paper (see Sect. 3.2).

In [6] layered graph transformation systems are considered, which are graph transformation systems where interleaving creation and deletion of edges with the same label is prohibited and creation of nodes is bounded. The paper shows such graph transformation systems are terminating.

The paper [10] simulates a graph transformation system by a Petri-net. Thus, the presence of edges with certain labels and the causal relationships between them are modeled, but no other structural properties of the graph. The paper uses typed graph transformation systems; thus, a type graph is used but, unlike in our weighted type graph method, it is fixed by the graph transformation system.

Finally, [3] was one of the inspirations for this paper. As shown in Sect. 3.3, its termination argument is subsumed by the weighted type graph technique.

7 Conclusion and Further Research

We introduced the weighted type graph technique for proving termination of graph transformation systems in the double pushout approach. The technique uses type graphs to assign weights to graphs that strictly decrease in each graph transformation step. It is simple and elegant and supports both uniform and non-uniform termination. Two simpler techniques, weighted edge and node counting (Sect. 3.2) and match bounds (Sect. 3.3) are subsumed by the technique.

Secondly, we showed that uniform termination of graph transformation systems of a specific form can be reduced to uniform termination of cycle rewriting, a form of rewriting related to string rewriting. This makes it possible to use the stronger termination algorithms of cycle rewriting for graph transformation systems. As a bonus, it provides an alternative proof of the undecidability of the termination problem of graph transformation systems.

Although all theorems have been stated and proved for (binary) multigraphs, a generalization to hypergraphs would be trivial. On the other hand, transferring the results to other graph transformation formalisms is harder. For example, in the single pushout approach, the graph transformation system corresponding to

the one of Ex. 1 is non-terminating, so the result of Ex. 2 (in which it is proved that this system is terminating) shows that the weighted type graph technique cannot be transferred one-to-one to single pushout graph transformation. It is left as future research to find similar arguments for the single pushout approach and other formalisms.

Another direction for further research is to allow for graph transformation systems with negative application conditions or more general application conditions. Note, however, that the implicit negative application condition of double pushout graph transformation, the dangling edge condition, can in some cases already be handled (see Ex. 2).

Finally, for interesting real-world applications, it would be interesting to generalize the technique to more expressive methods of specifying the initial graph languages, so that we can, for example, restrict to trees or rings of arbitrary size (both graph languages cannot be expressed by a type graph).

References

1. Uwe Aßmann. Graph rewrite systems for program optimization. *ACM Transactions on Programming Languages and Systems*, 22(4):583–637, 2000.
2. Paolo Bottoni, Kathrin Hoffman, Francesco Parisi Presicce, and Gabriele Taentzer. High-level replacement units and their termination properties. *Journal of Visual Languages and Computing*, 2005.
3. H.J. Sander Bruggink. Towards a systematic method for proving termination of graph transformation systems. In *Proceedings of GT-VC 2007*, 2007.
4. Andrea Corradini, Ugo Montanari, and Francesca Rossi. Graph processes. *Fundamenta Informaticae*, 26(3/4):241–265, 1996.
5. Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1: Foundations*. World Scientific, 1997.
6. Hartmut Ehrig, Karsten Ehrig, Juan de Lara, Gabriele Taentzer, Dániel Varró, and Szilvia Varró-Gyapay. Termination criteria for model transformation. In *Proceedings of FASE 2005*, volume 3442 of *LNCS*. Springer, 2005.
7. Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 15(3–4):149–171, 2004.
8. Adam Koprowski and Johannes Waldmann. Arctic termination ... below zero. In *Proceedings of RTA 2008*, volume 5117 of *LNCS*. Springer, 2008.
9. Detlef Plump. Termination of graph rewriting is undecidable. *Fundementa Informaticae*, 33(2):201–209, 1998.
10. Dániel Varró, Szilvia Varró-Gyapay, Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. Termination analysis of model transformations by petri nets. In *Proceedings of ICGT 2006*, volume 4178 of *LNCS*. Springer, 2006.
11. Hans Zantema. Termination of term rewriting by semantic labelling. *Fundementa Informaticae*, 24:89–105, 1995.
12. Hans Zantema, Barbara König, and H.J. Sander Bruggink. Termination of cycle rewriting. In *Proceedings of RTA-TLCA 2014*, volume 8560 of *LNCS*. Springer, 2014.