Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 175 (2007) 87–100
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Termination Criteria for DPO Transformations with Injective Matches

Tihamér Levendovszky[1]

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Hungary

Ulrike Prange[2] Hartmut Ehrig[2]

Department of Software Engineering and Theoretical Computer Science
Technical University of Berlin
Germany

Abstract

Reasoning about graph and model transformation systems is an important means to underpin model-driven software engineering, such as Model-Driven Architecture (MDA) and Model Integrated Computing (MIC). Termination criteria for graph and model transformation systems have become a focused area recently. This paper provides termination criteria for graph and model transformation systems with injective matches and finite input structure. It proposes a treatment for infinite sequences of rule applications, and takes attribute conditions, negative application conditions, and type constraints into account. The results are illustrated on case studies excerpted from real-world transformations, which show the termination properties of the frequently used "transitive closure" and "leaf collector" transformation idioms. An intuitive comparison with other approaches is also given.

Keywords: Termination Criteria, Graph Transformation, Model Transformation, DPO Approach

1 Introduction

Statements about termination of graph and model transformation systems have been proven recently, and a few transformation tools already support checking termination criteria [12]. This issue has mainly arisen for the following reason. When graph transformation is used for model transformation, the objective is to create an output mode either from the ground up or modifying existing models. If an output model must be achieved, a transformation must provide it within a finite number of

[1] Email: tihamer@aut.bme.hu
[2] Email: {uprange,ehrig}@cs.tu-berlin.de

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2007.04.019

steps. Therefore, examining the termination properties of the transformation can help to find an error in the model transformation. Taking into account that one of the most important applications of graph transformation is model transformation, well-developed termination criteria can be useful support for this application area.

When transforming a model, one or more input graph, a set of rules and constraints are available along with a control structure. The nontermination can be caused by the (i) input graph or (ii) the executed sequence of the rules. In the first case several examples can be constructed that illustrate nontermination. Assume a transformation rule takes an attribute of a node, and decrements it each time when the rule is fired. The rule has a constraint that it cannot be applied for zero attribute value. When infinity is allowed as the initial value of the attribute, this rule can be applied forever. A more obvious example is an input graph with infinite size. However, in practical model transformation applications, the input model is stored on a computer or on a distributed computer system. Therefore, assuming finite input graphs does not restrict the practical scope of the results.

The nontermination caused by a sequence of finite rules is more interesting for model transformation and its tool support. In this case the transformation either becomes stagnant or starts consuming the available system resources. An example for the stagnation case would be a transformation consisting of two rules executed in a loop after each other. The first rule creates an element, the second one deletes it. The transformation does not consume all the available system resources, but never stops. In our experience, the most prevalent reason is that the designer must have forgotten a constraint from the rules, and it is really useful to warn him of this fact. When a transformation needs a growing amount of system resources, the underlying reason can be twofold. (i) This transformation needs a stronger execution environment, or (ii) the transformation is nonterminating in nature, thus, there is no execution environment strong enough to perform this transformation. For instance, if a rule creates a node and can be executed exhaustively, it never stops creating nodes. Termination analysis can be a basis to prove that a stronger computational environment is needed, or the transformation suffers from an unintended side effect.

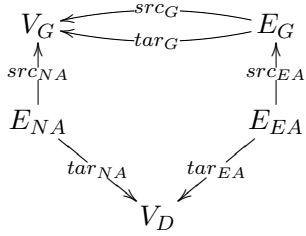
We use the formal framework of Adhesive High-Level Replacement (AHLR) Systems [6] applied to typed attributed graphs. We assume finite input structures and rules. Since this problem is algorithmically undecidable, we prove termination properties which can be used to examine the termination properties of the individual transformations analytically.

The main line of thought in this paper is as follows. The sequential rule applications are substituted with the composition of the rules. If one can show for the infinite rule sequences that the left-hand side of their composition tends to infinity, then the rule sequence terminates, since only finitely many elements are available in the start graph. This does not necessarily hold if one element in the rule can be matched to multiple elements in the host graph. Therefore, injective matches are assumed.

2 Backgrounds

Since we use the formalism and the results of the AHLR approach, we summarize the necessary definitions and results, based on [6]. In these definitions, we always mean typed, attributed graphs by mentioning graphs, which are defined as follows.

Definition 2.1 An E-graph $EG = (V_G, V_D, E_G, E_{NA}, E_{EA}, (src_j, tar_j)_{j\in\{G,NA,EA\}})$ consists of graph and data nodes V_G and V_D, and graph, node attribute and edge attribute edges E_G, $E_{NA}$and E_{EA}, respectively. The domains and codomains of the source and target functions src_j and tar_j for the corresponding edges E_j are depicted below.



Given a signature $DSIG = (S, OP)$ with attribute value sorts $S_D \subseteq S$, an attributed graph $AG = (EG, D)$ is an E-graph EG together with a $DSIG$-algebra D such that $V_D = \overset{\bullet}{\bigcup}_{s\in S_D} D_s$.

Given an attributed graph TG as type graph, a (typed attributed) graph $G = (AG, t)$ is an attributed graph AG together with a typing morphism $t : AG \to TG$.

Typed attributed graphs and the corresponding morphisms form the category $\mathbf{AGraphs_{ATG}}$.

We define a function to measure the size of a graph G.

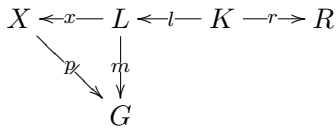
Definition 2.2 Given a graph $G = ((V_G, V_D, E_G, E_{NA}, E_{EA}, (src_j, tar_j)_{j\in\{G,NA,EA\}}), D)$, the size of G is denoted by $|G|$ and calculated as follows: $|G| = |V_G| + |E_G| + |E_{NA}| + |E_{EA}|$. G is finite if $|G| < \infty$.

We do not count the data nodes, since there may be infinitely many of them, but those relevant for the actual graph are linked by the attribute edges, which we do count. Moreover, the data part cannot be changed by applying a production.

Definition 2.3 A production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consists of finite graphs L, K and R, called left hand side, gluing graph and right hand side respectively, and two injective graph morphisms l and r that preserve the data part.

For practical purposes, it is important to restrict the applicability of a production by application conditions. In particular, we use negative application conditions, which forbid the existence of a certain subgraph.

Definition 2.4 A negative application condition of a production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is of the form $NAC(x)$, where $x : L \to X$ is an injective graph morphism. A graph morphism $m : L \to G$ satisfies $NAC(x)$ if there does not exist an injective graph morphism $p : X \to G$ with $p \circ x = m$.



Two graph productions (rules) are presented in Figure 1. The upper rule is applied first, as long as it can be matched against the input graph. A negative application condition ensures that at most one dashed arrow can be created between two vertices. The rule below "short-circuits" a dashed path with a length of two edges as long as possible. The resulted construct is referred to as transitive closure.

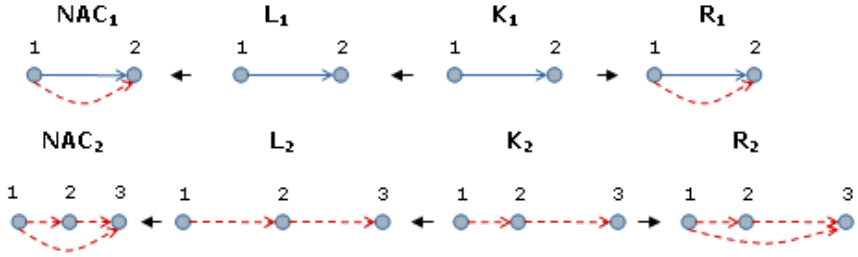
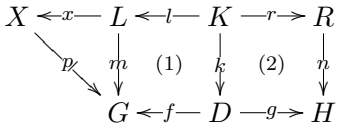


Fig. 1. Two productions computing the transitive closure

Definition 2.5 Given a graph production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and a graph G with a graph morphism $m : L \rightarrow G$, called match. If m satisfies all negative application conditions of p, a direct graph transformation $G \overset{p,m}{\Longrightarrow} H$ from G to a graph H is given by the following double pushout (DPO) diagram, where (1) and (2) are pushouts.



A sequence $G_0 \Rightarrow G_1 \Rightarrow ... \Rightarrow G_n$ of direct graph transformations is called a graph transformation and is denoted as $G_0 \overset{*}{\Rightarrow} G_n$. For $n = 0$ we have the identical graph transformation $G_0 \overset{id}{\Rightarrow} G_0$.

We say p is applicable to G via m, if m satisfies the NACs of p, pushouts (1) and (2) exist, and the resulting graph H satisfies additional constraints given by the system. In this paper we assume injective matches m and comatches n.

Definition 2.6 A graph transformation system $GTS = (P)$ consists of a set of graph productions P with or without negative application conditions. For a graph transformation system, there may be given a set of finite input graphs.

Remark 2.7 In Definition 2.6, we do not take arbitrary constraints into account. However, the results in this paper can treat all sorts of constraints, including those that are not formally defined, since their satisfaction is contained in the applicability

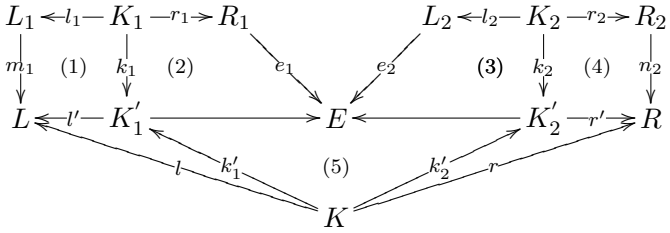
of a production, therefore, they have been integrated into the definition dealing with the applicability of a production, and thus, it appears in Definition 3.1.

Primarily, we need a definition for the termination of a graph transformation system. We extend the definition used in [10]. In [5], the treatment of a layering control structure is added to this definition. We extend the definition in such a way that an arbitrary control structure can be handled.

Definition 2.8 A graph transformation system $GTS = (P)$ terminates if there is no infinite sequence of direct graph transformations $G_0 \Rightarrow G_1 \Rightarrow ...$ applying rules from P starting from any input graph G_0, with respect to the control structure of the given graph transformation system.

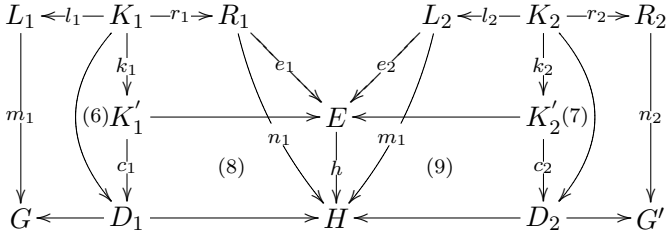
Up to now, the following definition of E-concurrent productions and the Concurrency Theorem have not been extended to productions with some kind of application conditions. Therefore we consider only plain productions in the following definition and theorem, as given in [6]. The results contributed in Section 3 are also valid when the rules contain negative application conditions.

Definition 2.9 Given two productions $p_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ and $p_2 = (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$, an E-dependency relation (E, e_1, e_2) is given by a graph E and injective morphisms $e_1 : R_1 \to E$, $e_2 : L_2 \to E$, which are jointly surjective. The E-concurrent production $p_1 *_E p_2$ is a production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ computed based on the following diagram, where double squares (1)(2) and (3)(4) form double pushouts, and (5) is a pullback. Note that the injectivity of e_1 and e_2 implies that of k_1, m_1, k_2, and n_2.



This definition can be applied recursively, using an E-concurrent production for p_1.

A transformation $G \overset{p_1,m_1}{\Longrightarrow} H \overset{p_2,m_2}{\Longrightarrow} G'$ is called E-related to $p_1 *_E p_2$ if there exist morphisms $h : E \to H$, $c_1 : K_1' \to D_1$ and $c_2 : K_2' \to D_2$ such that $h \circ e_1 = n_1$, $h \circ e_2 = m_2$, (6) and (7) commute and (8) and (9) are pushouts.

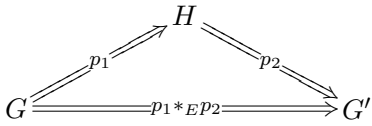


Theorem 2.10 (Concurrency Theorem) *Let (E, e_1, e_2) be an E-dependency relation for the productions p_1 and p_2 leading to the E-concurrent production $p_1 *_E p_2$.*

(i) *Synthesis: Given an E-related transformation sequence $G \Rightarrow H \Rightarrow G'$ via p_1 and p_2, then there is a synthesis construction leading to a direct transformation $G \Rightarrow G'$ via $p_1 *_E p_2$.*

(ii) *Analysis: Given a direct transformation $G \Rightarrow G'$ via $p_1 *_E p_2$, then there is an analysis construction leading to an E-related transformation $G \Rightarrow H \Rightarrow G'$ via p_1 and p_2.*

(iii) *Bijective correspondence: The synthesis and analysis constructions are inverse to each other up to isomorphism.*



3 A General Criterion for Injective Matches

In this section, we provide a general approach for termination within the scope of the DPO approach. These results also apply when the rules contain negative application conditions and other constraints.

Definition 3.1 An E-concurrent production p^* is an E-based composition if there is at least one input graph G_0 with an E-related transformation $G_0 \overset{p^*}{\Longrightarrow} H$.

This definition is required, because for the DPO approach, the definition of E-concurrent productions and the Concurrency Theorem have not been extended to handle negative application conditions and other constraints. Moreover, this definition guarantees, among others, that the constraints enforced by p_1 do not contradict to the constraints necessary for the application of p_2. Typical examples for constraints are attribute constraints, negative application conditions, type conformance for metamodels, constraints from the control flow branches, but other constructs are also possible.

In most of the practical cases it is simple to find such an input graph for an E-concurrent production, when there is no contradiction between the rules. Then, the left hand side of this rule is already an input graph, or it can be extended with regard to possible constraints.

An example for composing the bottom rule in Figure 1 with itself via a chosen E_1 is depicted in Figure 2.

Definition 3.2 Consider a possibly infinite sequence of graph productions p_i, $(i = 1, 2, ...)$ and a sequence of E-dependency relations $((E_i, e_i^*, e_{i+1}))$ leading to a sequence of their E-based compositions $(p_i^* = (L_i^* \leftarrow K_i^* \rightarrow R_i^*))$ with $p_1^* = p_1$ and $p_n^* = (p_1 *_{E_1} p_2) *_{E_2} ... *_{E_n} p_n$.

A cumulative LHS series of this sequence is the graph series L_n^* consisting of the left-hand side graphs of p_n^*. Moreover, a cumulative size series of a production

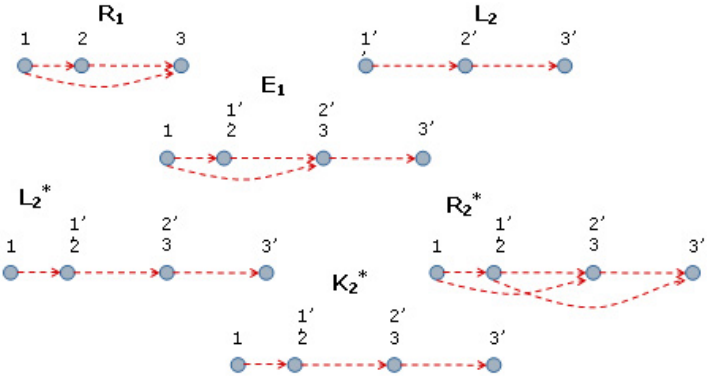


Fig. 2. An E-based composition of Rule 2 with itself

sequence is the nonnegative integer series $|L_n^*|$.

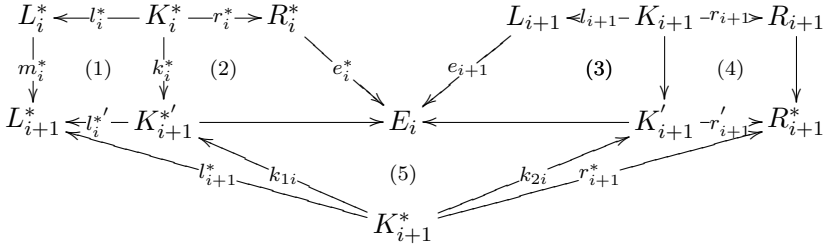
It is possible that there are several cumulative LHS series of a given production sequence, since, in general, two rules can be composed in different ways, choosing different E-dependency relations. For instance, if we want to compute p_3^* for our example, then we take the cumulative rule p_2^*, and compose it with the bottom rule from Figure 1. There are several possibilities to choose E_2: (i) we can short-circuit the path $1-2-3'$, (ii) $1-3-3'$, or (iii) R_2^* and L do not fully overlap. It is easy to see that in the first two cases L_3^* is isomorphic to L_2^*, but in the third case L_2^* must be extended to obtain L_3^*. However, there is no case, when L_3^* is smaller than L_2^*. If we consider injective matches only, this is true for the DPO approach in general.

Lemma 3.3 *The sequence $|L_i^*|$ (Def. 3.2) is monotonic nondecreasing. If $E_i \cong R_i^*$, L_i^* remains unchanged, thus, $|L_i^*| = \left|L_{i+1}^*\right|$. Otherwise, $L_i^* \ncong L_{i+1}^*$ and $|L_i^*| < \left|L_{i+1}^*\right|$, but L_{i+1}^* always contains an isomorphic subgraph of L_i^*.*

Proof. Since there is an injective morphism $m_i^* : L_i^* \to L_{i+1}^*$, we have $|L_i^*| \leq \left|L_{i+1}^*\right|$, and L_{i+1}^* contains an isomorphic subgraph of L_i^*.

Pushouts along isomorphisms are pullbacks, and pushouts and pullbacks are closed under isomorphism. Therefore, if $E_i \cong R_i^*$, we have isomorphisms k_i^* and m_i^*, which means that $L_i^* \cong L_{i+1}^*$ and $|L_i^*| = \left|L_{i+1}^*\right|$.

If $E_i \ncong R_i^*$, there are items $x \in E_i \backslash e_i^*(R_i^*)$, which have preimage in $K_{i+1}^{*'}$ but not in K_i^*, because (2) is a pushout. For (1) being a pushout, these items have to be added to L_i^* to obtain L_{i+1}^*, therefore $L_i^* \ncong L_{i+1}^*$, and $|L_i^*| < \left|L_{i+1}^*\right|$. □



Theorem 3.4 *A GTS = (P) (Def. 2.6)terminates if for all infinite cumulative LHS sequences (L_i^*) of the graph productions created from the members of P, it holds that*

$$\lim_{i\to\infty} |L_i^*| = \infty.$$

Note that we assume finite input graphs and injective matches.

Proof. We rely on the fact that if the constraints are satisfied, the E-based compositions are E-concurrent productions as well. Therefore, we can apply Theorem 2.10 for the topological part of the transformation, when we can assume that the constraints hold, which means the existence of the E-based composition.

In $\mathbf{AGraphs_{ATG}}$, Theorem 2.10 holds. Suppose there is an infinite transformation $G_0 \overset{p_1}{\Rightarrow} G_1 \Rightarrow \cdots$. Then there is a sequence of E-concurrent productions p_i^* leading to the transformations $G_0 \overset{p_i^*}{\Rightarrow} G_i$ (Theorem 2.10). All these productions are also E-based compositions with cumulative LHS series L_i^*. Since $\lim_{i\to\infty} |L_i^*| = \infty$, there exists an $N \in \mathbb{N}$ with $|G_0| < |L_N^*|$. But this means that there is no injective match $m_N^* : L_N^* \to G_0$, i.e. p_N^* is not applicable to G_0. □

The opposite direction of Theorem 3.4 does not hold in general, but for a finite number of input graphs. In this case, no infinite sequences of E-based compositions can be constructed.

Theorem 3.5 *If a GTS = (P) (Def. 2.6) terminates and we have only a finite number of input graphs up to isomorphism, then there are no infinite cumulative LHS sequences (L_i^*) of graph productions created from the members of P.*

Proof. Assume that *GTS* terminates and there is an infinite sequence (p_i^*) of E-based compositions.

For each p_i^* there exists an input graph G_i with an E-related transformation $G_i \overset{p_i^*, m_i}{\Longrightarrow} H_i$. Since there are only finite many input graphs, at least one of them has to appear infinitely many often. This means we have an input graph G with $\forall N \in \mathbb{N} \exists j > N : G \overset{p_j^*}{\Longrightarrow} H_j$. From Theorem 2.10 it follows that all p_i^* are applicable to G leading to an infinite transformation sequence. □

From Theorem 3.4 the next statement follows:

Lemma 3.6 *If $L_i^* \ncong L_{i+1}^*, \forall i$ for every cumulative LHS series (Def. 3.2), then the GTS terminates. If each graph appears only finitely many times in all cumulative LHS series, the GTS still terminates.*

Proof. Considering the first statement of the lemma, if two subsequent graphs in the cumulative LHS sequence are not isomorphic, they must grow in size because of Lemma 3.3. According to Theorem 3.4, this means that the GTS terminates.

Taking a cumulative LHS series at any position i, it grows in size within finite number of steps if there are only finite number of graphs in the series that are

isomorphic to L_i^*. The series must grow because of Lemma 3.3. Then we have $\lim_{i\to\infty} |L_i^*| = \infty$, and the GTS terminates because of Theorem 3.4. □

4 Case Studies

To show the practical relevance of the presented termination criteria, two case studies are provided. We take two transformation idioms from [1], and analyze their termination properties. Obviously, there are other proofs for these case studies, but we would like to illustrate how the technique contributed in this paper works for practical software model transformations, where the tool supports strict control flow constructs.

4.1 Transitive Closure

Using Theorem 3.4, we show that the transitive closure terminates. This is a frequently used transformation pattern. In case of variation of the 'class model to relational database management system (RDBMS) model' transformation [12] (also referred to as object-relational mapping), the traversal of the inheritance hierarchy and the association chains are performed using the transitive closure pattern.

Lemma 4.1 *The injective application of the transitive closure rule (the bottom rule in Figure 1) terminates for all finite input graphs.*

Proof. There are two cases. (i) When constructing E_k, it is not isomorphic to R_k^*. This means that in this case L_k^* must be extended to obtain L_{k+1}^* by Lemma 3.3. Therefore, in these steps, the cumulative LHS series grows. (ii) The other case needs more attention, since the cumulative LHS series does not grow in every step this time. We show that at a given stage of the transformation, this is possible finite times only. Suppose $E_k \cong R_k^*$ when constructing p_{k+1}^* from p_k^* and the original rule p. This leads only to a valid E-based composition if there is no dashed edge between $e_{k+1}(1)$ and $e_{k+1}(3)$ in E_k. In R_{k+1}^* no new nodes are added, but an additional edge (compared with R_k^*). Thus, after finite many steps we can only construct E-concurrency relations not isomorphic to the right hand side. This stems from the fact that the negative application condition forbids creating dashed edges between the nodes where there is one already. This means that an LHS can appear only a finite number of times in the cumulative LHS sequence, therefore, according to Lemma 3.6 the GTS terminates. □

4.2 Leaf Collector

The *LeafCollector* pattern is used to find the leaf elements in a tree structure. This idiom has been distilled from the transformation flattening a hierarchical data flow diagrams to a flat data flow representation [1]. In fact, *LeafCollector* does not modify the input graph, but finds a place where the next rule can be applied. Therefore, *LeafCollector* is a useful idiom of many software model transformations, and it is worth examining its termination properties.

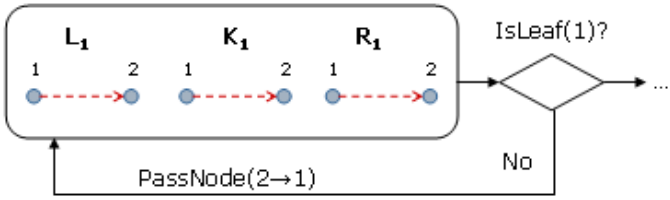


Fig. 3. The Leaf Collector Transformation Idiom

A possible formulation of the pattern is depicted in Figure 3. This idiom is particularly interesting, because it strongly builds on a sophisticated control structure of the transformation tool. The diamond in the figure can be implemented in several ways. In GReAT [1], it is implemented as test rule, whereas it is a branch condition in VIATRA [13] and VMTS [11]. The other required feature is parameter passing. This means that host graph nodes and edges matched in one of the previous rules can be passed to a subsequent rule. The matching algorithm considers these elements already bound. This can accelerate the matching process, and facilitates the separation of complex rules. If there are no passed parameters, the matching algorithm starts to match with unbound elements. In our example, the rule is bound to any of the suitable places in the input graph on the first execution. On the subsequent runs, the graph node matched to the rule node 2 is passed to the rule node 1. Therefore, the matching algorithm finds a node adjacent to the one passed as a parameter. The output of this idiom is *node*1 when it is a leaf. Therefore, *node*1 is passed further along the branch where the ellipses are depicted. The parameter passing mechanism is implemented with different syntax in the aforementioned tools, thus, we focus on the notion only without formalizing it. From the mathematical point of view, this construct is modeled as a restriction on the possible E-based compositions.

Since the idiom is obviously not concerned with self-loops, injective matches are assumed. Then we compute the E-based composition of the rule with itself. In this case it is rather simple, because the parameter passing reduces the number of the possible E-dependency relations to one.

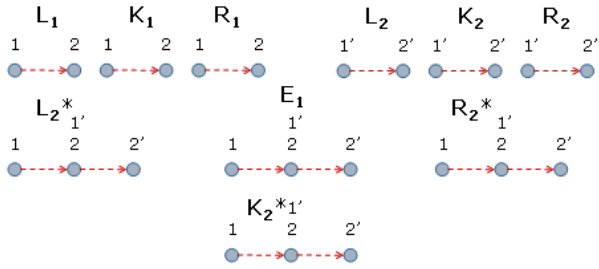


Fig. 4. E-based Composition for Leaf Collector - Acyclic Case

Lemma 4.2 *The transformation LeafCollector (depicted in Figure 3) terminates if and only if the input graph does not contain a directed cycle.*

Proof. Firstly, we compute the E-based composition of the rule with itself.

Because of the parameter passing, the E_i is created as follows: R_i^* contains only one node n_i that is a target of an incoming edge and it is not a source of any outgoing edge. Then the node $1^{(i)}$ in L_{i+1} is mapped to n_i in E_i, the others are mapped to different vertices and edges.

$1^{(i)}$ in L_{i+1} can either be mapped to an E_i element that is not mapped to any R_i^* element, or otherwise. Based on that, there are two possible categories of E-dependency relations. The first option is depicted in Figure 4 for p_2^*. Since there are only control conditions, it is the same as the E-concurrent production, where the E_1 is determined by the parameter passing. The control structure limits the number of the composed productions only. Obviously, L_i^*, K_i*, and R_i^* are the same, because the rule does not change the input graph: it searches for a specific element.

Pushouts along isomorphisms are pullbacks, and pushouts and pullbacks are closed under isomorphism. Therefore, $E_i \cong L_{i+1}^*$. Thus, L_i^* is a directed path consisting of i edges. This means that in this case the transformation terminates according to Theorem 3.4.

When $1^{(i)}$ in L_{i+1} is mapped to an E_i element that is mapped to any R_i^* element, it automatically creates a directed cycle. An example for this structure is depicted in Figure 5. In this case it is possible that we have a nonincreasing cumulative size series. According to Theorem 3.5, it is possible that this structure does not terminate. □

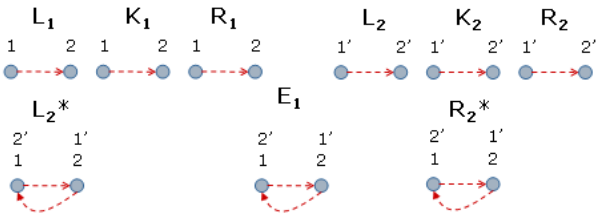


Fig. 5. E-based Composition for Leaf Collector - Cyclic Case

According to Lemma 4.2, if the input graph does not contain directed cycles, the transformation terminates, otherwise it is possible that the transformation does not terminate. In practice, this condition can be guaranteed in model transformation systems. (i) Most of the modeler tools offer a containment hierarchy, and along this hierarchy it is ensured by the tool that there are no directed cycles. (ii) Directed cycles in inheritance hierarchy causes semantical problems, it may also be forbidden by the tool.

If there are no such constraints in the model, the *LeafCollector* should be extended with additional construct in order to avoid nontermination. A possible solution is to add an *isProcessed* attribute to the nodes, which is *false* by default, and set by the rule if it is matched. Another solution is to introduce helper edges between the processed nodes, and introduce NACs to forbid the match at the same place again.

This case study illustrates that with the proposed termination analysis method, we could obtain the structure that causes the nontermination. Therefore, this tech-

nique is suitable for constructive analysis besides the decision issues.

5 Related Work

In [2], termination criteria have been developed for graph rewriting applied to program transformation. The criteria aim at this specific problem domain. The approach assumes that there can be no parallel edges with the same labels between two nodes. This leads to a termination criteria for specific (edge-accumulative) rules if the label and node sets are finite. Moreover, subtractive rules are investigated, which are conceptually similar to deletion layers examined in [5]. These results assume more restricted types of rules, compared to those analyzed in this paper.

In [3], a theory has been developed for the DPO approach. It provides abstract termination criteria by a measure function F. The paper also shows concrete termination criteria such as the number of nodes, the number of edges. Based on this assumption, it proves termination criteria for other control structures. However, these criteria are violated in the second case study with respect to the concrete criteria of edge and node numbers. However, no explicit relationship has been established between the proposed definition of a termination criterion and the notion that the transformation stops within a finite number of steps.

In [5], results have been developed for layered grammars. These results formalize and extend the contributions provided in [7] [4]. The provided criterion ensures that the creation of all objects of a type should precede the deletion of the object of this type. Therefore, a layer deleting an object of a given type cannot create such an object, nor the subsequent rules. This means that the productions in a deletion layers terminate for the reasons detailed above if the types are taken into consideration.

A nondeletion layer cannot contain rules that delete a node. It is ensured by a negative application condition that a rule cannot be applied twice at the same match. Furthermore, if a rule creates an object of a given type, it is not allowed to match any object of that type in that or any subsequent layers. Since Layer 0 uses the finite input graph, and there cannot be a match at the same place, and the rules in Layer 0 cannot create elements of a type whose instances they match, the rules can be executed only a finite number of times. The next layer terminates for similar reasons: it can only use elements of a type whose instances have already been created. Since Layer 0 has terminated, Layer 1 is passed a finite graph, thus, the situation is similar to that in case of Layer 0.

In our context, this means that only a finite fully overlapping ($E_i \cong R_i^*$) sequences are possible, since the the NAC forbids the E-based composition at a given position more than once. Otherwise $|L_i^*|$ must increase. Unfortunately, there are situations, where these criteria do not hold. In our first case study, the second rule matches and creates an element of the same type.

The methods discussed as related work are not restricted to injective matches as opposed to our approach.

6 Conclusions

A novel contribution of this paper is to provide termination criteria for general productions allowing recursion within the scope of DPO and typed attributed graph transformation, assuming injective matches. This can be a theoretical basis to prove that certain control flows of rules are terminating, where the other - algorithmically underpinned - criteria cannot be applied. In general, however, it is hard to find all the possible sequences of graph productions, and prove that the corresponding series $|L_i^*|$ exceeds all limits. This is expected, since the termination of a GTS is undecidable [10]. However, the stricter and the more deterministic the ordering of the rules is, the higher is the chance that we can deal with the sequences. For example, in the tool Visual Modeling and Transformation System (VMTS) [11], the control structures are as strict as possible, and nondeterminism is avoided if possible. Moreover, parameter passing between the rules (external causalities) decrease the number of the possible E_k graphs, since the nodes and edges connected by a morphism from R_k^* to L_{k+1} must be mapped to the same nodes and edges in E_k. We have also contributed two case studies, which solve the termination issue of two frequently used transformation idioms called "transitive closure" and "leaf collector".

Another contribution is that in the composition of the productions in Definition 3.1, attributes are also considered, and the proposed method is open to other constraint specification approaches. Furthermore, it regards control structures and parameter passing.

Future work includes the extension of these results to noninjective matches. Furthermore, constraint checking to decide whether a composition rule exists is not simple in the general case, when not only the attributes set by the transformation steps are considered. Also, we would like to analyze more idioms and frequent building blocks. A library of building blocks with proven termination properties may help the tools to overcome the algorithmic undecidability. Since where the algorithms fail, the structural investigation can offer a solution.

7 Acknowledgments

The activities described in this paper were supported, in part, by the SegraVis Training Network and by the National Office for Research and Technology (Hungary).

References

[1] Agrawal, A., Vizhanyo, A., Kalmar, Z., Shi, F., Narayanan, A., Karsai, G: Reusable Idioms and Patterns in Graph Transformation Languages 2004. Proc. 2nd International Workshop on Graph Based Tools (GraBaTs 2004). Satellite workshop of ICGT 2004, Rome, Italy, 2004.

[2] Assmann, U., Graph rewrite systems for program optimization, ACM TOPLAS 22, 2000, pp. 583-637

[3] Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G.: "Termination of High-Level Replacement Units with Application to Model Transformation", VLFM 2004, Electronic Notes of Theoretical Comp.Sci. (ENTCS) vol.127, no.4 (2005), Elsevier, pp. 71-86.

[4] Bottoni, P., Taentzer, G., Schuerr, A. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. In Proc. Visual Languages 2000 IEEE Computer Society. pp.: 59-60.

[5] Ehrig, H., Ehrig, K., de Lara, J., Taentzer, G., Varró, D., Varró-Gyapay,Sz.: "Termination Criteria for Model Transformation", FASE 2005, LNCS, pp. 49-63.

[6] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: "Fundamentals of Algebraic Graph Transformation", EATCS Monographs in Theoretical Computer Science, Springer, 2006

[7] de Lara, J., Taentzer, G. 2004. Automated Model Transformation and its Validation with AToM3 and AGG. In DIAGRAMS2004 (Cambridge, UK). Lecture Notes in Artificial Intelligence 2980, pp.: 182198. Springer.

[8] Lengyel, L., Levendovszky, T., Charaf, H.: "Eliminating Crosscutting Constraints from Visual Model Transformation Rules", ACM/IEEE 7th International Workshop on Aspect-Oriented Modeling, Montego Bay, Jamaica, October 2, 2005.

[9] Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: "A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS", Electronic Notes in Theoretical Computer Science, International Workshop on Graph-Based Tools (GraBaTs) Rome, 2004.

[10] Plump, D.: "Termination of graph rewriting is undecidable", Fundamenta Informaticae, 33(2):201209, 1998

[11] VMTS Web Site, http://avalon.aut.bme.hu/~tihamer/research/vmts

[12] Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varró, D., Varró-Gyapay, Sz.: Model Transformation by Graph Transformation: A Comparative Study,ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, 2005

[13] Varró, D.: Automated Model Transformations for the Analysis of IT Systems. PhD thesis, Budapest University of Technology and Economics, Department of Measurement and Information Systems (2004)