Termination Analysis of Model Transformations by Petri Nets *

Dániel Varró[1], Szilvia Varró–Gyapay[1], Hartmut Ehrig[2], Ulrike Prange[2], and Gabriele Taentzer[2]

[1] Budapest University of Technology and Economics
Department of Measurement and Information Systems
{varro, gyapay}@mit.bme.hu
[2] Technical University of Berlin
{ehrig, uprange, gabi}@tfs.cs.tu-berlin.de

Abstract. Despite the increasing relevance of model transformation techniques in model-driven software development, research is mainly conducted to the specification and the automation of such transformations. However, since the transformations themselves may also contain conceptual flaws, it is essential to formally analyze them prior to executing them on user models. In the current paper, we focus on a central validation problem of trusted model transformations, namely, *termination* and propose a Petri net based analysis method that provides a sufficient criterion for the termination problem of model transformations captured by graph transformation systems.

Keywords: graph transformation, termination, model transformation, Petri nets.

1 Introduction

Many researchers and practitioners have recently revealed that model driven software development relies not only on the precise definition of modeling languages taken from different domains, but also on the unambiguous specification of transformations between these languages. To provide a standardized support for capturing queries, views and transformations (QVT) between modeling languages defined by their standard MOF metamodels, the Object Management Group (OMG) is soon to issue QVT [17] as a standard. QVT provides a declarative rule-based, model transformation language where control structures are restricted to embedding transformation rules into each other.

Graph transformation (GT) [8,20] has been applied successfully to many model transformation (MT) problems. Many success stories were in the field of model analysis which aim at projecting high-level UML models into mathematical domains by model transformations to carry out formal analysis.

* This work was partially supported by the Segravis Research Training Network. Dániel Varró was also supported by the János Bolyai Scholarship.

A. Corradini et al. (Eds.): ICGT 2006, LNCS 4178, pp. 260–274, 2006.

As revealed in a recent study [14], graph transformation and QVT-like declarative techniques show a close correspondence. A first precise formulation of this correspondence has been studied in [19]. As a consequence the theoretical background of graph transformation is expected to provide useful results for QVT.

Problem statement. A core problem, which is very vaguely addressed by QVT, is related to the *correctness of model transformations*, namely, to guarantee that certain semantic properties hold for a *trusted model transformations*. For instance, when transforming UML models into mathematical domains, the results of a formal analysis can be invalidated by erroneous model transformations as the systems engineers cannot distinguish whether an error is in the design or in the transformation. In case of QVT, it is possible that the embedded transformation rules interfere with each other and thus they may cause semantic problems, which is not acceptable for trusted model transformations.

Most typical correctness properties of a trusted model transformation are termination, uniqueness (confluence) and behaviour preservation. In [11], we proposed a set of sufficient criteria that guarantees the termination of model transformations specified by so-called layered graph transformation systems (GTS). While this technique was applicable to various practical model transformation problems, further experiments have revealed that these sufficient criteria exclude model transformations where rules are causally dependent on themselves.

Objectives and Approach. In the current paper, we provide a Petri Net based technique for the termination analysis of model transformations specified by GTSs. As termination is undecidable for graph grammars in general [18], we propose a sufficient criterion, which either proves that a GTS is terminating, or it yields a "maybe nonterminating" (do not know) answer.

The essence of our technique is to derive a simple Petri net which simulates the original GTS by abstracting from the structure of instance models (graphs) and only counting the number of elements of a certain type. If we manage to prove by algebraic techniques that the Petri net runs out of tokens in finitely many steps regardless of the initial marking, then we can conclude that the original GTS is terminating due to simulation. In order to handle graph transformation systems with negative application conditions as well, we introduce the notions of forbidden and permission patterns, and overapproximate how different rules influence each other when generating permissions.

As the derived Petri net model is of managable size (comparable to the number of elements in the metamodels), our technique can yield positive results for judging the termination of various model transformation problems captured by graph transformation techniques.

*Structure of the paper.*The rest of the paper is organized as follows: Sec. 2 presents a running example where we specify (with graph transformation rules) a transformation from UML class diagrams to relational databases. Sec. 3 provides an overview on graph transformation systems and Place / Transition (P/T) nets. Sec. 4 proposes a P/T net abstraction of GTS with rules having negative application conditions (NAC). Sec. 5 presents sufficient conditions for termination of

GTSs by solving algebraic inequalities. Sec. 6 discusses related work and finally Sec. 7 presents our conclusions and proposals for future work.

2 Motivating Example: The Object-Relational Mapping

As the motivating example of the current paper, we map simple UML class diagrams into relational database tables by using one of the standard solutions. This transformation problem (with several variations) is frequently used as a model transformation benchmark of high practical relevance [15].

The source and target languages (UML and relational databases, respectively) are captured by their corresponding metamodels in Fig. 1. In Sec. 3.1, metamodels will be represented formally by means of type graphs [9], while instance models will be graphs typed over a type graph.

UML class diagrams in our paper consist of classes arranged into an inheritance hierarchy (by *parent* edges). Classes have attributes (*attrs*), which are typed over classes (*type*). Directed associations are leading from a source (*src*) class to a destination (*dst*) class.

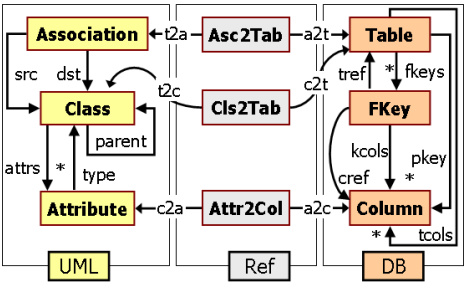


Fig. 1. Metamodels (type graphs): Source, reference, target

Relational databases consist of tables, which are composed of columns (*tcols*). Each table has a single primary key column (*pkey*). Foreign key (*FKey*) constraints can be assigned to tables (*fkeys*). A foreign key refers to certain columns (*cref*) of a table (*tref*), and it is related to the columns *kcols* of (local) referencing table.

These metamodels (adapted from [15]) are extended by a *reference metamodel* to interconnect the elements of the source and the target language. This way it defines the main guidelines of (this variant of) the object-relational mapping itself, which can be summarized as follows:

- Each top-level UML class (i.e. a top-most class in the inheritance tree) is projected into a database table. Two additional columns are derived automatically for each top-level class: one for storing a unique identifier (primary key), and one for storing the type information of instances.
- Each attribute of a UML class will appear as columns in the table related to the top-level ancestor of the class. For the sake of simplicity, the type of an attribute is restricted to user-defined classes. The structural consistency of valid object instances in columns is maintained by foreign key constraints.
- Each UML association is projected into a table with two columns pointing to the tables related to the source and the target classes of the association by foreign key constraints.

3 Introduction to Graph Transformation and Petri Nets

Now we provide a brief overview on the formal background of graph transformation and Petri nets. Only those concepts will be introduced which are essential for presenting our main results in Sec. 4 and 5.

3.1 Typed Graph Transformation

Type and Instance Graphs. The metamodels of different modeling languages are frequently formalized as type graphs and instance models are typed over this type graph. The traditional instance-of relationship between metamodels and models is captured formally by a typing morphism.

A *graph* $G = (N, E, src, trg)$ is a 4-tuple with a set N of nodes, a set E of edges, a source and a target function $src, trg : E \to N$. A *type graph* TG is an ordinary graph. An *instance graph* G is typed over TG by a typing morphism $type : G \to TG$. Let $card(G, x)$ denote the *cardinality* (i.e. the number of graph objects) of a type $x \in TG$ in graph G. Formally, $card(G, x) = |\{n \mid n \in N \cup E \wedge type(n) = x\}|$.

For the current paper, we assume that there is a unique edge of a certain type between two nodes, i.e., if $src(e_1) = src(e_2) \wedge trg(e_1) = trg(e_2) \wedge type(e_1) = type(t_2) \Rightarrow e_1 = e_2$, which simplifies the proofs of our theorems.

Graph Transformation. Graph transformation (GT) [8] provides a rule-based manipulation of graph models. A *graph transformation rule* $r = (L \xleftarrow{l} K \xrightarrow{r} R)$ typed over a type graph TG is given by triple where L (left-hand side, LHS), K (context) and R (right-hand side, RHS) graphs are typed over TG and graph morphisms l, r are injective and assumed to be type preserving.

The *negative application conditions* (NACs) of a GT rule are a (potentially empty) set of pairs (N, n) with N being a graph also typed over TG and $n : L \to N$ being an injective graph morphism. A GT rule with NACs is denoted shortly as $r = (L \xleftarrow{l} K \xrightarrow{r} R, \{L \xrightarrow{n_i} N^i\})$ $(i = 1 \ldots k)$. Moreover, we assume that no rules exist where all L and N are empty.

Application of a Rule. The *application* of a rule to a *host model graph* G alters the model graph by replacing the pattern defined by L with the pattern defined by R. This is performed by (i) *finding an injective matching* $m : L \to G$ of the L pattern in model graph G; (ii) *checking the negative application conditions* N which prohibit the presence of certain model elements, i.e. for each NAC $n : L \to N$ of a rule no injective graph morphism $q : N \to G$ exists with $m = q \circ n$; (iii) *removing* a part of the model graph M that can be mapped to L but not to R yielding an intermediate graph D; (iv) *adding* new elements to the intermediate graph D which exist in R but not in L yielding the derived graph H. A GT step is denoted formally as $G \overset{r,m}{\Longrightarrow} H$, where r and m denote the applied rule and the matching along which the rule was applied, respectively. In the paper, we follow the *Double Pushout Approach* [8].

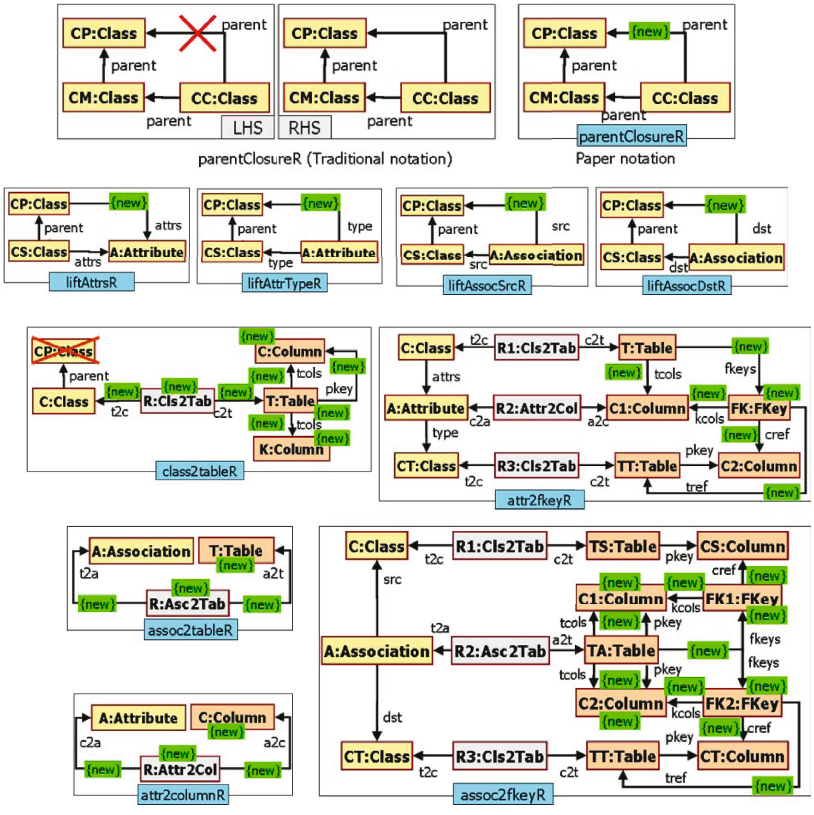


Fig. 2. Model transformation from UML to relational databases

Example 1. A sample graph transformation rule calculating the transitive closure of the parent relation is depicted in the top rule (*parentClosureR*) of Fig. 2. The rule prescribes that if class *CP* is parent of class *CM* (i.e. there is a *parent* edges between them), and *CM* is a parent of class *CC*, but there is no *parent* edge from *CC* to *CP*, then such an edge should be created.

For a more compact presentation of the rules, we abbreviate the L, N and R graphs of a rule into one, and we only mark the (images of) graph elements to be removed (*del*), or created (*new*). We assume that all elements in R marked as *new* are implicitly present in the negative application condition N as well. In case of rule *class2TableR* we use crossed lines to denote the second negative application condition (that is not part of R).

Example 2. The object-relational mapping is captured by the set of graph transformation rules in Fig. 2. The entire transformation starts with a preprocessing phase when the transitive closure of *parent* relations is calculated (*parentClosureR*), and then all attributes and associations are lifted up to the top-level classes in the inheritance (*parent*) hierarchy (rules *liftXYZ*). Then the main model

transformation (Fig. 2) proceeds as described in Sec. 2 by transforming classes into tables (*class2tableR*), associations into tables (*assoc2tableR*), attributes into columns (*attr2columnR*), attribute types and destination class of associations into foreign key constraints (*attr2fkeyR* and *assoc2fkeyR*).

A *graph transformation system* $GTS = (R, TG)$ consists of a type graph TG and a finite set R of graph transformation rules typed over TG. A *graph grammar* $GG = (GTS, G_0)$ consists of a graph transformation system $GTS = (R, TG)$ and a so-called *start (model) graph* G_0 typed over TG.

The *state space* $Sem(GG)$ generated by a graph grammar $GG = (GTS, G_0)$ is defined as a graph where nodes are model graphs, and edges are graph transformation steps $G \overset{r,m}{\Longrightarrow} H$ such that the source and target nodes of the edge are graphs G and H, respectively. Starting from G_0 the *state space* (i.e. the reachable model graphs) of the GG is represented taking into account all applicable rules from a given model graph for all possible matchings.

A *graph grammar* $GG = (G_0, GTS)$ is *terminating* if there are no infinite sequences of rule applications starting from G_0. A *graph transformation system* $GTS = (R, TG)$ is called *terminating* if for all G_0, the corresponding graph grammar $GG = (G_0, GTS)$ is terminating.

3.2 Place/Transition Nets

In the current section we give a short introduction into the theory of Place/Transition nets based on [16].

A *Place/Transition net* (or shortly P/T net) is a 4-tuple $PN = (P, T, E, w)$ where P is a set of *places* (represented graphically as circles), T is a set of *transitions* (represented as horizontal bars), $E \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs* (where no arcs connect two places or two transitions), and the *weight function* $w : E \to \mathbb{N}^+$ maps arcs to positive integers.

Places may contain tokens. The distribution of tokens at different places is called a *marking* $M : P \to \mathbb{N}$, which maps places to non-negative integers. The initial marking is denoted as M_0.

The token distribution can be changed in the net by firing transitions. A transition t is *enabled* (i.e. it may fire), if each of its input places contain at least as many tokens as it is specified by the weight function. The *firing* of an enabled transition t removes a $w(p, t)$ amount of tokens from the input places, and $w(t, p)$ tokens are produced on each output place p. As a result, the marking M changes to M' (denoted as $M \overset{t}{\Longrightarrow} M'$) according to $\forall p \in P : M'(p) = M(p) - w(p, t) + w(t, p)$.

The *incidence matrix* W of a (finite) net describes the net token flow (of the P/T net) when firing a transition. Mathematically, W is a $|P| \times |T|$–dimensional matrix of non-negative integers $\mathbb{N}$ such that $w_{ij} = w(t_j, p_i) - w(p_i, t_j)$, where $1 \leq i \leq |P|, 1 \leq j \leq |T|$.

After firing a transition t in marking M, the result marking M' can be computed with the incidence matrix: $M' = M + W \cdot \underline{e}_t$, where $\underline{e}_t$ is a $|T|$-dimensional unit vector, where the t-th component is 1 and the others are 0.

A *(transition) firing sequence* $s = \langle t_1, t_2, \ldots, \rangle$ is a sequence of transition firings starting from state M_0 such that $M_0 \stackrel{t_1}{\Longrightarrow} M_1, \stackrel{t_2}{\Longrightarrow} \ldots$, i.e. for all $1 \leq j$ t_j is enabled in M_{j-1} and M_j is yielded by the firing of t_j in M_{j-1}.

The marking of the net after executing the first k steps of the firing sequence s can be calculated by the state equation: $M_k = M_0 + W \cdot \underline{\sigma}$, where $\underline{\sigma}$ is the *transition occurrence vector* or *Parikh–vector* of the trajectory s counting the number of occurrences of individual transitions in the firing sequence.

4 A Petri Net Abstraction of Graph Transformation

4.1 Definition of the Core Abstraction

First we map a graph transformation system *without negative application* conditions into a Petri net (which is called *cardinality (P/T) net* in the sequel) by only keeping track of the number of objects in the instance graph (separately for each node and edge in the type graph) but abstracting from the structure of the instance graph.

Informally speaking, since the LHS of a GT rule requires the presence of nodes and edges of certain types, the derived transition removes tokens from all the places storing the instances of the corresponding types. Furthermore, the RHS of a GT rule guarantees the presence of nodes and edges of certain types, thus the derived transition generates tokens for the places storing the instances of such types. Later we show that this is a proper abstraction, i.e. the derived P/T net simulates the original GTS, i.e. when a GT rule is applicable, the corresponding transition in the P/T net can be fired as well.

This mapping $\mathcal{F}(GTS) = (\mathcal{F}_{TG}, \mathcal{F}_G, \mathcal{F}_R) \rightarrow PN$ (where $GTS = (R, TG)$ and $PN = (P, T, E, w)$ with initial marking M_0) is formally defined as follows:

- $\mathcal{F}_{TG} : TG \rightarrow P$*: Types into places.* For each node and edge $y \in N_{TG} \cup E_{TG}$ in the type graph TG, a corresponding place $p_y = \mathcal{F}(y)$ is defined in the cardinality P/T net.
- $\mathcal{F}_G : G \rightarrow M_0$*: Instances into tokens.* For each node and edge $x \in N_G \cup E_G$ in an instance graph G with type $y = type(x)$, a token is generated in the corresponding marking $M_G = \mathcal{F}(G)$ of the target P/T net. Formally, for all places $p_y = \mathcal{F}(y)$, the marking of the net is defined as $M_G(p_y) = card(G, y)$.
- $\mathcal{F}_R : R \rightarrow (T, E, w)$*: Rules into transitions.* For each rule r in the graph transformation system GTS, a transition $t_r = \mathcal{F}(r)$ is generated in the cardinality P/T net such that
  - *Left-hand side:* If there is a graph object x in L with $y = type(x)$, then an incoming arc (p_y, t_r) is generated in the P/T net where $p_y = \mathcal{F}(y)$ and the weight of the arc $w(p_y, t_r)$ is equal to the number of graph objects in L of the same type y. Formally, if $\forall x, y : x \in L \wedge y = type(x) \wedge \mathcal{F}(y) = p_y \Rightarrow (p_y, t_r) \in E \wedge w(p_y, t_r) = card(L, y)$.
  - *Right-hand side:* If there is a graph object x in R with $y = type(x)$, then an outgoing arc (t_r, p) is generated in the P/T net where $p_y = \mathcal{F}(y)$ and the weight of the arc $w(t_r, p_y)$ is equal to the number of graph objects in R of the same type y. Formally, if $\forall x, y : x \in R \wedge y = type(x) \wedge \mathcal{F}(y) = p_y \Rightarrow (t_r, p_y) \in E \wedge w(t_r, p_y) = card(R, y)$.

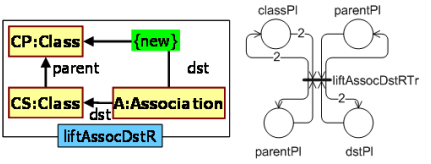


Fig. 3. Transition corresponding to rule liftAssocDstR

In Fig. 3 rule *liftAssocDstR* of the example in Fig. 2 is shown on the left together with the corresponding transition *liftAssocDstR* (on the right) of the P/T net abstraction of the example. Note that indices of $\mathcal{F}()$ will be omitted for simplicity.

As the GT rule *liftAssocDstR* contains two *Class* nodes, one *Association* node, one *parent* edge and one *dst* edge, the corresponding transition is enabled if the corresponding type places (with identical labels) contain at least 2, 1, 1, and 1 tokens, respectively. Since the application of the rule preserves all items and creates one *dst* edge, the firing of transition *liftAssocDstR* puts 2, 1, 1, and 2 tokens to these places, respectively.

Note, however, that the transition of Fig. 3 is always enabled and thus, it would directly cause non-termination. Therefore, we now extend our abstraction technique to handle graph transformation rules with negative application conditions as well, which are frequently used in model transformation problems.

4.2 Extensions for Negative Conditions

Permission Places. In order to cope with NACs, the P/T net is extended with so-called *permission places* to restrict the firing of a transition. We add one permission place for each NAC in the GTS, and the idea of a permission place is to count how many times the GT rule can be applied to the current instance graph (such that the corresponding NAC does not violate these matchings).

- *Start graph.* The initial marking of permission places shall enable the firing of a transition as many times as the corresponding GT rule is applicable to the start graph by giving a permission token.
- *Removing permissions.* If a new matching of some NAC N_i of a GT rule r is generated or an existing matching of the LHS of the same rule r is destroyed by the application of some GT rule r' then one or more tokens should be removed from the permission place corresponding to N_r^i.
- *Creating permissions.* If an existing matching of the NAC of a GT rule r is destroyed or a new matching of the LHS of the same rule r is generated by the application of some GT rule r' then one or more tokens should be generated to the permission place corresponding to N_r^i.

Unfortunately, the exact number of tokens created for or removed from a permission place depends on the actual graph structure. Therefore, we cannot derive a constant weight *a priori* for the corresponding arcs in the P/T net; instead we write $w(G)$ on such arcs to denote that the weight of the arc is dependent on graph G. However, we know that such an arc weight $w(G)$ is finite, i.e. we can only generate and remove a finite number of new permissions for any permission place.

Overapproximation for Permissions. Therefore, we need to define an overapproximation of the potential number of rule applications, which still simulates the GTS,

yet it is precise enough to detect termination for a certain class of model transformation problems.

- In our proposal, we only remove one token from a permission place when it is absolutely guaranteed (by analyzing the original GT rule) that a *permission should be destroyed* each time the rule is applied. In case of GT rules with NAC, such a situation is when a GT rule cannot be applied on the same matching twice due to a NAC.
- In case of *generating a permission*, we should consider all possible values for the arc weight $w_i(G)$, thus we create a new variable c_i which runs over positive integers.

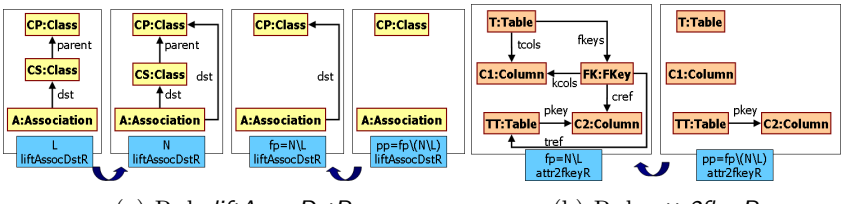
Permission and Forbidden Patterns. An initial idea for granting permissions is to consider the causalities of GT rules, i.e. when a rule generates a new matching for another rule, a new permission is generated as well. However, this solution is unable to handle cases when GT rules are generating a bounded number of new matchings for themselves (i.e., when a rule is causally dependent on itself).

For instance, each application of rule *liftAssocDstR* (in Fig. 3) generates a new *dst*, thus a new matching for itself, which seems to be a direct cause for non-termination. On the other hand, if the meaning of a permission is related to the number of *Class-Association* pairs not connected by a *dst* edge, we notice that this number is strictly decreasing, thus no new permission is granted by GT rule *liftAssocDstR* for itself. This insight is captured formally by *forbidden and permission patterns*.

Definition 1 (Forbidden and permission pattern). *Let* $GTS = (R, TG)$ *be a graph transformation system . A* forbidden pattern fp_r^i *is defined for each NAC* N_r^i *of rule* r *as the smallest subgraph of* N_r^i *that contains* $N_r^i \setminus L_r$ *(also called as the* context *of* $n^i : L_r \to N_r^i$*).*

The permission pattern pp_r^i *(of the same NAC* N_r^i*) is defined as smallest subgraph of* fp_r^i *that contains* $N_r^i \setminus L_r$ *(also called as the* boundary *of* $n_r^i : L_r \to N_r^i$*), which is defined formally as* $fp_r^i \setminus (N_r^i \setminus L_r)$.

Informally, the permission pattern can be interpreted as an LHS pattern having a NAC with the forbidden pattern. The exact number of permissions for a rule is calculated as the number of matchings of the permission pattern having the forbidden pattern as a NAC.



(a) Rule *liftAssocDstR* (b) Rule *attr2fkeyR*

Fig. 4. Forbidden and permission patterns

Example 3. The concepts of forbidden and permission patterns are demonstrated in Fig. 4(a). The forbidden pattern (*FP*) of rule *liftAssocDstR* contains a *dst* edge leading from Association *A* to Class *CP*. Here $N \setminus L$ contains the single *dst* edge while the two nodes are added to guarantee that the forbidden pattern forms a graph. In order to obtain the permission pattern (*PP*), we simply remove this *dst* edge from the forbidden pattern.

Definition of cardinality P/T with permission places. The *cardinality P/T net with permission places* of GTS is a $PN = (P, T, E, w)$ derived by the mapping $\mathcal{F}_{pp}(GTS)$ by extending $\mathcal{F}(GTS)$ in the following way:

- *Variables as weight functions.* We extend the weight function of a P/T net to $w : E \to \mathbb{N}^+ \cup V$ where V is a set of variables ranging over $\mathbb{N}^+$.
- *NACs into permission places.* For each NAC N^i of a rule r a corresponding permission place $p_{r_{N^i}} = \mathcal{F}_{pp}(r_{N^i})$ is defined in the cardinality net.
- *Matchings of permission patterns into tokens (initial marking).* For each NAC N^i of a rule r as many tokens are generated in the corresponding permission place as the number of injective matchings m of permission pattern pp_r^i in the instance graph G which satisfies the derived NAC $pp_r^i \to fp_r^i$, (i.e., there is no injective matching of the forbidden pattern fp_r^i to G along m).
- *NACs into pre arcs.* For each rule r with NACs $N^1, \ldots, N^k$, if there is an injective morphism $k_i : N^i \to R$ compatible with r for some NAC N^i (informally, everything included in the NAC N^i exists or it is created by the RHS), an incoming arc $(p_{r_{N^i}}, t_r)$ is generated in the P/T net with weight 1.
- *Rule actions into post arcs.* For each pair of rules $r = (L_r \xleftarrow{l} K_r \xrightarrow{r} R_r)$ with NACs $N^1, \ldots, N^k$ and $r' = (L'_r \xleftarrow{l} K'_r \xrightarrow{r} R'_r)$, an outgoing arc $(t_{r'}, p_{r_{N^i}})$ $(i : 1 \leq i \leq k)$ is generated in the P/T net (i.e. from the transition of rule r' to the permission place of r_{N^i}) with a *variable* arc weight $v_{r',r_{N^i}}$ if
  1. at least one graph object o is deleted by r' (from the forbidden pattern fp_r^i of r) such that there exists a graph object $o' \in N^i \setminus L_r$, and $type(o) = type(o')$ or
  2. at least one graph object o is created by r' such that there exists a graph object $o' \in pp_r^i$, and $type(o) = type(o')$.

Informally, instead of regarding the causality between two rules based upon the RHS of rule r' and the LHS of r, we define causality between the effects of a rule r' and the permission pattern of r.

Furthermore, in order to overapproximate the graph dependent arc weights $w(G)$, we introduce variables as weights for such arcs. As a consequence, for each step of the P/T net, we can substitute the variables with proper values to simulate the original GTS in a step-wise way. In order to prove termination later in Sec. 5, we will show that any substitution of these variables fulfill certain algebraic properties.

The *incidence matrix* of the P/T net abstraction of GTS with NACs is denoted as $W(\underline{v})$, which notation emphasizes that W contains variables at locations where new permissions are generated for a rule.

	UML								Ref								
	Assoc	Class	Attr	src	dst	attrs	type	par	t2a	t2c	c2a	A2T	C2T	A2C	a2t	c2t	a2c
parentClosureR	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
liftAttrsR	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
liftAttrTypeR	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
liftAssocSrcR	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
lisftAssocDstR	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
class2tableR	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0
assoc2tableR	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0
attr2columnR	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1
attr2fkeyR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
assoc2fkeyR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	DB									permission places (for each NAC)										
	Tab	FKey	Col	tref	fkeys	pkey	tcols	cref	kcols	parCl	lAttr	lAttrT	lAssSrc	lAssDst	cl2ta1	cl2ta2	as2tab	attr2c	att2fk	ass2fk
parentClosureR	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
liftAttrsR	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0
liftAttrTypeR	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
liftAssocSrcR	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0
lisftAssocDstR	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
class2tableR	1	0	2	0	0	1	2	0	0	0	0	0	0	0	0	-1	0	0	v_1	v_2
assoc2tableR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	v_3	v_4
attr2columnR	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	v_5	v_6
attr2fkeyR	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	-1	0
assoc2fkeyR	0	2	2	1	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	-1

Fig. 5. Incidence matrix of the P/T net abstraction

The incidence matrix of the example GTS is given in Fig. 5. The places (columns) refer to the type places corresponding to the type graph of Fig. 1, while transitions (rows) refer to corresponding rules of Fig. 2. The right-most columns of the matrix denote permission places. Note that the incidence matrix is independent of the initial marking of the cardinality P/T net, thus our termination result is valid for any initial marking.

It is worth pointing out that the proposed abstraction highly relies on the fact that a RHS contains at least one of its NACs. Note that this is typical for model transformation problems where NACs are frequently used to prevent the application of a rule multiple times on the same matching.

Example 4 (Cardinality P/T net with permission patterns). Rule *parentClosureR* generates new *parent* edges, which are required for the matching of rule *liftAssocDstR* (see Fig. 2), thus the two rules are causally dependent. However, no new permissions are generated for the latter, since rule *parentClosureR* should remove a *dst* edge (see the forbidden pattern) or create new *Class* or *Association* nodes (see the permission pattern) for a new permission to be generated (see the permission and forbidden patterns in Fig. 4(a)).

On the other hand, rule *class2tableR* generates new permissions for rule *attr2fkeyR*, since the tables created by the former are present in the permission pattern of the latter (which consists of tables *T*, *TT* and columns *C1* and *C2*, see Fig. 4(b)). Consequently, a variable v_1 is used as the weight of the corresponding arc leading from the transition of *class2table* to the permission place of *attr2fkeyR*.

5 Termination Analysis of Graph Transformation

Now we propose a termination analysis for GTS using a generalization of non-repetitiveness results from P/T nets [16].

A P/T net is *partially repetitive* if there exists a marking M_0 and a firing sequence s from M_0 such that some transition occurs infinitely many times in s. Furthermore, a main result from P/T net theory states that a P/T net with the incidence matrix W is partially repetitive if and only if there exists a Parikh–vector $\underline{\sigma} \geq \underline{0}, \underline{\sigma} \neq \underline{0}$ such that $W^T \cdot \underline{\sigma} \geq \underline{0}$. As a consequence, if a P/T net is not partially repetitive (i.e., no Parikh–vector $\underline{\sigma} \geq \underline{0}, \underline{\sigma} \neq \underline{0}$ exists that satisfies $W^T \cdot \underline{\sigma} \geq \underline{0}$),

then only finite firing sequences exist from any intial marking M_0, which proves termination.

Our generalization lies in the fact we do not require the existence of the incidence matrix W. Instead we state that if sequences of state vectors fulfill the condition that at least one component of the state vector is decreasing (wrt. each previous state vector in the sequence) in each step it guarantees that the $\underline{0}$ state is reached in finite steps. Our reason for this generalization is that W may contain variables at permission places.

Theorem 1. *If for all infinite sequences* $\{M_i\} = M_0, M_1, \ldots$ *of n-dimensional (state) vectors of nonnegative integer values with* $M_j - M_{j-1} < \infty$ *for all* j

$$(1) \quad \forall i, \forall j : j > i, M_i \not\equiv \underline{0} \Rightarrow \exists k : M_j[k] - M_i[k] < 0, \textit{and}$$

$$(2) \quad \forall i, \forall j : j > i, M_i \equiv \underline{0} \Rightarrow M_j \equiv \underline{0}$$

then $M \equiv \underline{0}$ *in finitely many steps, i.e.* $\exists s : M_s \equiv \underline{0}$ *(where* $M_j[k]$ *denote component* k *in vector* M_j*).*

Then, we claim that mapping $\mathcal{F}()$ is a proper abstraction in the sense that the derived P/T net *without permission places* simulates the original GTS . In other terms, whenever a rewriting step is executed in the GTS on an instance graph, then the corresponding transition can always be fired in the corresponding marking in the P/T net, furthermore, the result marking is an abstraction of the result graph.

Theorem 2 (Cardinality P/T net simulates GTS). *Let* $GTS = (R, TG)$ *be a graph transformation system and* $PN = (P, T, E, w)$ *be a cardinality P/T net derived by the mapping* $\mathcal{F}(GTS)$*. Furthermore, let* G, H *be instance graphs typed over* TG*. Then* PN *simulates* GTS*, formally*

$$\forall G, H, r, o : (G \overset{r,o}{\Longrightarrow} H) \Rightarrow (M_G \overset{t_r}{\Longrightarrow} M_H),$$

where $\mathcal{F}(G) = M_G$*,* $\mathcal{F}(H) = M_H$*, and* $\mathcal{F}(r) = t_r$*.*

Finally, as a termination "oracle", we solve quadratic inequalites based on the incidence matrix of the P/T net with variables as defined in Sec. 4.1-4.2. If there are no solutions for the inequality for any evaluation of variables in the incidence matrix, we state that the original GTS is terminating.

Theorem 3 (Termination). *Let* $W(\underline{v})$ *be the incidence matrix of a cardinality P/T net* $PN = \mathcal{F}_{pp}(GTS)$ *derived as the abstraction of a* GTS*.*

If $\exists \underline{\sigma} \exists \underline{v} \quad W(\underline{v}) \cdot \underline{\sigma} \geq \underline{0}$ *has no solutions with* $\underline{v} \geq 1, \underline{\sigma} \geq \underline{0}, \underline{\sigma} \neq \underline{0}$ *(thus* $\forall \underline{\sigma} \forall \underline{v} \quad \exists k : (W(\underline{v}) \cdot \underline{\sigma})[k] < 0$*), then* GTS *is terminating.*

In order to show that the quadratic inequality $W(\underline{v}) \cdot \underline{\sigma} \geq \underline{0}$ has no solutions for proving the termination of GTSs with negative application conditions, we used a symbolic optimization toolkit (GAMS [12]) which supports mixed integer nonlinear programming.

6 Related Work

Relation of Graph Transformation and Petri Nets. The main idea of this paper is to analyze graph transformation systems via Petri nets. In fact, there is a long tradition concerning the relationship of both areas. The basic observation is that a P/T net is essentially a rewriting system on multisets, which allows to encode the firing of P/T nets as a direct graph transformation in the Double Pushout approach using discrete graphs and empty interfaces for the productions only (see [7]). Taking into account general graphs and nonempty interfaces graph transformation systems are closer to some generalizations of Petri nets, like contextual nets. This relationship has been used in [2] to model concurrent computations of graph grammars.

Vice versa the existence of powerful analysis techniques for P/T nets motivates to simulate graph transformation by P/T nets [3], which allows to conclude correctness properties of graph grammars from properties of corresponding P/T nets. The main novelty of this paper wrt. [3] (and subsequent papers of the authors) is that (i) we take into account also negative application conditions of graph transformations and (ii) the size of the derived P/T is dependent on the type graph and not to the instance graph. The price we have to pay for a more efficient termination analysis is that our P/T net can be too abstract to verify all the safety properties investigated in [3].

Termination of Graph Transformation Systems. Termination of graph transformation systems is undecidable in general [18], but several approaches have been considered to restrict a graph transformation system such that termination can be shown. The classical approach of proving termination is to construct a monotone function that measures graph properties, and to show that the value of such a function decreases with every rule application. Concrete criteria such as the number of nodes and edges of certain types have been considered by Aßman in [1]. However, he sticks to these concrete criteria, while Bottoni et.al. [5] developed a general approach to termination based on measurement functions.

With respect to termination for graph transformation systems, the current work generalizes and formalizes the work begun at [13]. This, in fact, is an extension of the layering conditions for deleting grammars proposed in [6], which were used for parsing. A main advantage of our approach with respect to the termination requirements of this parsing algorithm is that we do not require to partition the rules (and the alphabet) into layers.

As pointed out already in the introduction, we have presented termination criteria for graph transformation systems in [11], which allow to prove termination of several practical relevant model transformations. However these criteria are not applicable to model transformations where rules are causally dependent on themselves (e.g. transitive closure) like our motivating example. Since each layer of [11] can be treated separately by our current techniques, furthermore, the termination criteria proposed in [11] imposes a special structure on the derived incidence matrix of the P/T net, it is possible to show that our termination analysis technique based on P/T nets subsumes our former results in [11].

7 Conclusion

In this paper, we have presented a termination analysis technique for model transformations expressed as graph transformation systems using an abstraction into Petri nets. This way, the termination problem of (a special class of) graph transformation systems can be proved by its Petri net abstraction using algebraic techniques. Since the termination of graph transformation systems is undecidable in general, our approach yields a sufficient criterion: either it proves that a GTS is terminating, or gives a "do not know" answer.

We believe that our results can also be useful for proving the termination of QVT-based model transformations, which also uses a very limited set of control structure. For instance, triple graph grammars (TGG) [21] provide a declarative means to specify model transformations, and show a strong conceptual correspondence with bidirectional QVT mappings. Moreover, a pair of traditional (operational) graph transformations can be easily derived for each TGG rule, and then our termination criteria become directly applicable.

Although not mentioned explicitly, the termination criteria presented can also be used for graph transformation with node type inheritance, since a flattening to graph transformation without inheritance is available in [4]. Thus, the termination analysis can always be done and need not be translated back.

Acknowledgments. The authors are grateful to András Pataricza for initial ideas, and Paolo Baldan for fruitful discussions. Valuable comments of anonymous reviewers were also highly helpful.

References

1. Aßmann, U. 2000. *Graph Rewrite Systems for Program Optimization.* ACM TOPLAS, vol. 22(4), pp. 583–637, ACM Press, New York.
2. Bardohl, P., 2000. *Modelling Concurrent Computations: From Contextual Petri Nets to Graph Grammars.* PhD thesis, University of Pisa.
3. Baldan, P., Corradini, A., and König, B. 2001. *A Static Analysis Technique for Graph Transformation Systems* In *Proc. CONCUR 2001*, LNCS 2154, pp. 381–395. Springer.
4. Bardohl, R., Ehrig, H., de Lara J., and Taentzer, G. 2004. *Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation.* In *Proc. FASE'04*, LNCS 2984, pp. 214–228. Springer.
5. Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G. 2004. *Termination of High-Level Replacement Units with Application to Model Transformation.* In proceedings of VLFM'04, ENTCS.
6. Bottoni, P., Taentzer, G., Schürr, A. 2000. *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.
7. Corradini, A. 1996 *Concurrent Graph and Term Graph Rewriting.* In Proc. CONCUR'96, LNCS 1119, pp. 438–464, Springer.
8. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., and Löwe, M. *In [20]*, chap. Algebraic Approaches to Graph Transformation — Part I: Basic Concepts and Double Pushout Approach, pp. 163–245. World Scientific, 1997.

9. Corradini, A., Montanari U., Rossi, F. *Graph Processes.* Fundamenta Informaticae 26(3/4):241-265.
10. Ehrig, H., and Ehrig, K., and Prange, U., and Taentzer, G. Fundamentals of Algebraic Graph Transformation. *Monographs in Theoretical Computer Science. An EATCS Series*, Springer-Verlag New York, Inc., 2006.
11. Ehrig, H., Ehrig, K., de Lara, J., Taentzer G., Varró, D., Varró-Gyapay, Sz. *Termination Criteria for Model Transformation.* In FASE 2005: Internation Conference on Fundamental Approaches to Software Engineering (Edinburgh, UK), LNCS 3442, pp. 49-63, Springer, 2005.
12. GAMS: General Algebraic Modeling System. `http://www.gams.com`.
13. de Lara, J., Taentzer, G. 2004. *Automated Model Transformation and its Validation with AToM^3 and AGG.* In DIAGRAMS'2004 (Cambridge, UK). Lecture Notes in Artificial Intelligence 2980, pp.: 182–198. Springer.
14. Küster, J., Sendall, S., Wahler, M.. *Comparing two model transformation approaches.* In OCL and Model Driven Engineering 2004.
15. Model Transformations in Practice (Satellite Workshop of MODELS 2006) `http://sosym.dcs.kcl.ac.uk/events/mtip`.
16. T. Murata. Petri nets: Properties, analysis and applications. In *Proc. IEEE*, vol. 77, pp. 541–580. 1989.
17. Object Management Group. *QVT: Request for Proposal for Queries, Views and Transformations.* `http://www.omg.org`.
18. Plump, D. 1998. *Termination of Graph Rewriting is Undecidable.* Fundamenta Informaticae 33(2):201-209.
19. Rensink, A., and Nederpel R. 2006. Graph transformation semantics for a QVT language. In *Proc. Fifth Intern. Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2006)*, ENTCS, pp. 45–56. Elsevier. In Press.
20. Rozenberg, G. (ed) 1997. *Handbook of Graph Grammars and Computing by Graph Transformation.* World Scientific. Volume 1.
21. Schürr, A. Specification of Graph Translators with Triple Graph Grammars. In *Proc. WG94: Int. Workshop on Graph-Theoretic Concepts in Computer Science*, vol. 903 of *LNCS*, pp. 151–163. Springer-Verlag, 1994.