Technische Universität Berlin

Forschungsberichte der Fakultät IV – Elektrotechnik und Informatik

Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation: Long Version

Fernando Orejas, Esther Guerra,
Juan de Lara, Hartmut Ehrig

Bericht-Nr. 2009-9
ISSN 1436-9915

Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation (Long Version)

Fernando Orejas[1], Esther Guerra[2], Juan de Lara[3], and Hartmut Ehrig[4]

[1] Universitat Politècnica de Catalunya (Spain), orejas@lsi.upc.edu
[2] Universidad Carlos III de Madrid (Spain), eguerra@inf.uc3m.es
[3] Universidad Autónoma de Madrid (Spain), jdelara@uam.es
[4] Technische Universität Berlin (Germany), ehrig@cs.tu-berlin.de

Abstract. Model-to-model (M2M) transformation consists in transforming models from a source to a target language. Many transformation languages exist, but few of them combine a declarative and relational style with a formal underpinning able to show properties of the transformation. Pattern-based transformation is an algebraic, bidirectional, and relational approach to M2M transformation. Specifications are made of patterns stating the allowed or forbidden relations between source and target models, and then compiled into low level operational mechanisms to perform source-to-target or target-to-source transformations. In this paper, we study the compilation into operational triple graph grammar rules and show: (i) correctness of the compilation of a specification without negative patterns; (ii) termination of the rules, and (iii) completeness, in the sense that every model considered relevant can be built by the rules.

1 Introduction

Model-to-model (M2M) transformation is an enabling technology for recent software development paradigms, like Model-Driven Development. It consists in transforming models from a source to a target language and is useful, e.g. to migrate between language versions, to transform a model into an analysis domain, and to refine a model. In some cases, after performing the transformation, the source and target models can be modified separately. Therefore, it is useful to be able to execute transformations both in the forward and backward directions to recover consistency. Thus, an interesting property of M2M transformation languages is to allow specifying transformations in a direction-neutral way, from which forward and backwards transformations can be automatically derived.

In recent years, many M2M specification approaches have been proposed [1, 2, 13–17] with either *operational* or *declarative* style. The former languages explicitly describe the operations needed to create elements in the target model from elements in the source, i.e they are unidirectional. Instead, in declarative approaches, a description of the mappings between source and target models is provided, from which *operational mechanisms* are generated to transform in the forward and backward directions.

In this paper, we are interested in declarative, bidirectional M2M transformation languages. Even though many language proposals exist, few have a formal basis enabling the analysis of specifications or the generated operational mechanisms [18]. In previous work [3], we proposed a new graphical, declarative, bidirectional and formal approach to M2M transformation based on triple patterns. Patterns specify the allowed or forbidden relations between two models and are similar to graph constraints [6], but for triple graphs. The latter are structures made of three graphs representing the source and target models, as well as the correspondence relations between their elements. Thus, in pattern-based transformation we define the set of valid pairs of source and target models by constraints, and not by rules. Then, patterns are compiled into operational rules working on triple graphs to perform forward and backward transformations.

In the present work, we prove certain properties of the compilation of pattern-based specifications into rules. First, we show that our compilation mechanism generates graph grammars that are terminating. This result is interesting as it means that we do not need to use external control mechanisms for rule application [11]. Second, we prove that the transformation rules are sound with respect to the positive fragment of the specification. This means that a triple graph satisfies all positive patterns in a specification if and only if it is terminal with respect to the generated rules. In other words, the operational mechanisms actually do their job, and this corresponds to the notion of *correctness* in [18]. Finally, we also prove completeness of the rules, i.e. that the rules are able to produce any model *generated* by the original M2M specification. These generated graphs are a meaningful subset of all the models satisfying the specification.

Altogether, we think that this work paves the way to using formal methods in one of the key activities of Model-Driven Development: the specification and execution of M2M transformations. The rest of the paper has been organized as follows. Section 2 provides an introduction to triple graphs and to the kind of transformation rules that we use in this paper. Section 3 introduces M2M pattern specifications, their syntax and semantics. Section 4 is the core of the paper: first we introduce the transformation rules associated to a pattern specification and then we prove their termination, soundness and completeness. Then, in Section 5 we compare the approach that we follow with some other approaches to M2M transformation. Finally, in Section 6 we draw some conclusions and we sketch some future work. In addition, along the paper we use a small running example describing the transformation of class diagrams into relational schemas [16]. An appendix includes the full proofs for all the presented results.

2 Preliminaries

This section introduces the basic concepts that we use throughout the paper about triple graphs and triple graph transformation. Triple graphs [17] model the relation between two graphs called source and target through a correspondence graph and a span of graph morphisms. In this sense, if we consider that models are represented by graphs, triple graphs may be used to represent transformations, as well as transformation information through the connection graph.

Definition 1 (Triple Graph and Morphism). *A triple graph* $G = (G_S \overset{c_S}{\leftarrow} G_C \overset{c_T}{\rightarrow} G_T)$ *(or just* $G = \langle G_S, G_C, G_T \rangle$ *if* c_S *and* c_T *may be considered implicit) consists of three graphs* G_S*,* G_C*, and* G_T*, and two morphisms* c_S *and* c_T*. A triple graph morphism* $m = (m_S, m_C, m_T)\colon G^1 \rightarrow G^2$ *is made of three graph morphisms* m_S*,* m_C *, and* m_T *such that* $m_S \circ c_S^1 = c_S^2 \circ m_C$ *and* $m_T \circ c_T^1 = c_T^2 \circ m_C$*.*

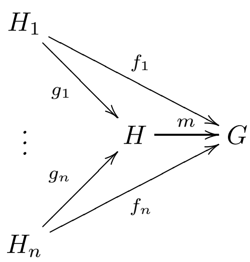
Given a triple graph G, we write $G|_X$ for $X \in \{S, T\}$ to refer to a triple graph whose X-component coincides with G_X and the other two components are the empty graph, e.g. $G|_S = \langle G_S, \emptyset, \emptyset \rangle$. Similarly, given a triple graph morphism $h\colon G_1 \rightarrow G_2$ we also write $h|_X\colon G_1|_X \rightarrow G_2|_X$ to denote the morphism whose X-component coincides with h_X and whose other two components are the empty morphism between empty graphs. Finally, given G, we write i_G^X to denote the inclusion $i_G^X\colon G|_X \rightarrow G$, where the X-component is the identity and where the other two components are the (unique) morphism from the empty graph into the corresponding graph component.

Triple graphs form the category **TrG**, which can be formed as the functor category $\mathbf{Graph}^{\cdot\leftarrow\cdot\rightarrow\cdot}$. In principle, we may consider that **Graph** is the standard category of graphs. However, the results in this paper still apply when **Graph** is a different category, as long as it is an adhesive-HLR category [12, 6] and satisfies the additional property of n-factorization (see below). For instance, **Graph** could also be the category of typed graphs or the category of attributed (typed) graphs.

Definition 2 (Jointly surjective morphisms). *A family of graph morphisms* $\{H_1 \overset{f_1}{\rightarrow} G, \ldots, H_n \overset{f_n}{\rightarrow} G\}$ *is* jointly surjective *if for every element* e *(a node or an edge) in* G *there is an* e' *in* H_k*, with* $1 \leq k \leq n$ *such that* $f_k(e') = e$*.*

A property satisfied by graphs and by triple graphs, is n-factorization, a generalization (and also a consequence) of the property of pair factorization [6]:

Proposition 1 (n-factorization). *Given a family of graph morphisms* $\{H_1 \overset{f_1}{\rightarrow} G, \ldots, H_n \overset{f_n}{\rightarrow} G\}$ *with the same codomain* G*, there exists a graph* H*, a monomorphism* m *and a jointly surjective family of morphisms* $\{H_1 \overset{g_1}{\rightarrow} H, \ldots, H_n \overset{g_n}{\rightarrow} H\}$ *such that the diagram below commutes:*

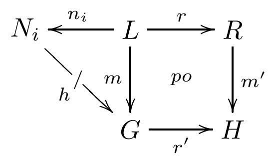


It may be noticed that if a category satisfies the n-factorization property and $\{f_1, \ldots, f_n\}$ in the above diagram are monomorphisms then so are $\{g_1, \ldots, g_n\}$.

In this paper, a graph transformation rule is a monomorphism $(L \xrightarrow{r} R)$, possibly equipped with some Negative Application Conditions (NACs) in its left-hand side for limiting its application. The reason is that in our approach we just need to use *non-deleting rules*. Hence, rule application is defined by a pushout. Moreover, in our case, when applying a rule to a given graph G, it is enough to consider the case where the morphism that matches L to G is a mono:

Definition 3 (Non-Deleting Triple Rule, Rule Application, Terminal Graph). *A (non-deleting) triple rule $p = \langle N, L \xrightarrow{r} R\rangle$, consists of a triple monomorphism r and a finite set of negative application conditions $N = \{L \xrightarrow{n_i} N_i\}_{i \in I}$, where each n_i is a triple monomorphism.*

A monomorphism $m\colon L \to G$ is a match for the rule $p = \langle N, L \xrightarrow{r} R\rangle$ if m satisfies all the NACs in N, i.e. for each NAC $L \xrightarrow{n_i} N_i$ there is no monomorphism $h\colon N_i \to G$ such that $m = h \circ n_i$. Given a match $m\colon L \to G$ for p, the application of p to G via m, denoted $G \Rightarrow_{p,m} H$, is defined by the pushout below:



where m' is called the comatch of this rule application.

Given a set TR of transformation rules, a triple graph G is terminal for TR if no rule in TR can be applied to G.

3 Pattern-Based Model-to-Model Transformation

Triple patterns are similar to graph constraints [6, 9]. We use them to describe the allowed and forbidden relationships between the source and target models in a M2M transformation. In particular, we consider two kinds of patterns. Negative patterns, which are denoted by just a triple graph, describe relationships that should not occur between the source and target models. This means that, from a formal point of view, negative patterns are just like negative graph constraints, i.e. a (triple) graph G satisfies the negative pattern N if N is not a subgraph of G (up to isomorphism). Positive patterns specify possible relationships between source and target models. Positive patterns consist of a set of negative premises and a conclusion. As we can see below, satisfaction of P-patterns does not coincide exactly with satisfaction of graph constraints.

Definition 4 (Patterns, M2M Specification). *An N-pattern, denoted $N(Q)$ consists of a triple graph Q. A P-pattern $\mathcal{S} = \{N(Q \xrightarrow{n_j} C_j)\}_{j \in J} \Rightarrow Q$ consists of*

- *The conclusion, given by the triple graph Q.*

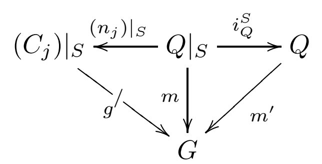
– *The negative premises* $N(Q \xrightarrow{n_j} C_j)$*, given by the inclusions* $Q \xrightarrow{n_j} C_j$*.*

An M2M specification SP *is a finite set of P and N-patterns. Given a specification* SP*, we denote by* SP^+ *(respectively,* SP^-*) the positive fragment of* SP*, i.e. the set of all P-patterns in* SP *(respectively, the set of all N-patterns in* SP*).*

A P-pattern is intended to describe a class of transformations denoted by triple graphs. P-patterns describe, simultaneously, source-to-target and target-to-source transformations. In this sense, there are two notions of satisfaction associated to P-patterns: forward satisfaction, associated to source-to-target transformations and backward satisfaction, associated to target-to-source transformations. Then, a triple graph G forward satisfies a pattern $\{N(Q \xrightarrow{n_j} C_j)\}_{j\in J} \Rightarrow Q$ if whenever Q_S is embedded in the source of G (and the premises are forward satisfied by the embedding), Q is embedded in G. And an embedding m of Q_S in G_S forward satisfies a premise $N(Q \xrightarrow{n_j} C_j)$ if there is no embedding m' of $(C_j)_S$ in G_S such that m' extends m. Backward satisfaction is the converse notion.

Definition 5 (Pattern Satisfaction).

– *A monomorphism* $m\colon Q|_S \to G$ *forward satisfies a negative premise* $N(Q \xrightarrow{n_j} C_j)$*, denoted* $m \models_F N(Q \xrightarrow{n_j} C_j)$ *if there does not exist a monomorphism* $g\colon (C_j)|_S \to G$ *such that* $m = g \circ (n_j)|_S$*. Similarly,* $m\colon Q|_T \to G$ *backward satisfies a negative premise* $N(Q \xrightarrow{n_j} C_j)$*, denoted* $m \models_B N(Q \xrightarrow{n_j} C_j)$ *if there does not exist a monomorphism* $g\colon (C_j)|_T \to G$ *such that* $m = g \circ (n_j)|_T$*.*
– *A triple graph* G *forward satisfies a P-pattern* $\mathcal{S} = \{N(Q \xrightarrow{n_j} C_j)\}_{j\in J} \Rightarrow Q$*, denoted* $G \models_F \mathcal{S}$*, if for every monomorphism* $m\colon Q|_S \to G$*, such that, for every* j *in* J*,* $m \models_F N(Q \xrightarrow{n_j} C_j)$*, there exists a monomorphism* $m'\colon Q \to G$ *such that* $m = m' \circ i_Q^S$*:*



– *A triple graph* G *backward satisfies a P-pattern* $\mathcal{S} = \{N(Q \xrightarrow{n_j} C_j)\}_{j\in J} \Rightarrow Q$*, denoted* $G \models_B \mathcal{S}$*, if for every monomorphism* $m\colon Q|_T \to G$*, such that, for every* j *in* J*,* $m \models_B N(Q \xrightarrow{n_j} C_j)$*, there exists a monomorphism* $m'\colon Q \to G$ *such that* $m = m' \circ i_Q^T$*.*
– G *satisfies* $\mathcal{S}$*, denoted* $G \models \mathcal{S}$*, if* $G \models_F \mathcal{S}$ *and* $G \models_B \mathcal{S}$*.*
– *A triple graph* G *satisfies an N-pattern* $N(Q)$*, denoted* $G \models N(Q)$ *if there is no monomorphism* $h\colon Q \to G$*.*

Though the abuse of notation, given a monomorphism $m\colon Q \to G$, we may also say that m forward satisfies a negative premise if the monomorphism $i_G^S \circ m|_S\colon Q|_S \to G$ forward satisfies it.

Example 1. The left of Fig. 1 shows a specification describing a transformation between class diagrams and relational schemas [16]. When considered in the forward direction, the transformation creates a database schema to store in tables the attributes of the classes, where classes of the same inheritance hierarchy are mapped to the same table. The first pattern `C-T` states that top-level classes (i.e., those without parents in the inheritance hierarchy) are mapped to tables. Note that we use a notation similar to UML object diagrams (i.e. `c:C` represents a node `c` of type `Class`). `N(NoDup)` is an N-pattern that forbids associating two tables to the same class. `A-Co` and `ChC-T` are P-patterns with an empty set of premises. `A-Co` says that attributes of a class are stored in columns of the associated table. Finally, `ChC-T` specifies that children and parent classes are mapped to the same table. The right of the same figure shows an example of satisfaction. Graph G satisfies all patterns in the specification and, in particular, the diagram shows how an occurrence of $\texttt{ChC-T}|_{\texttt{S}}$ is extended to `ChC-T`.

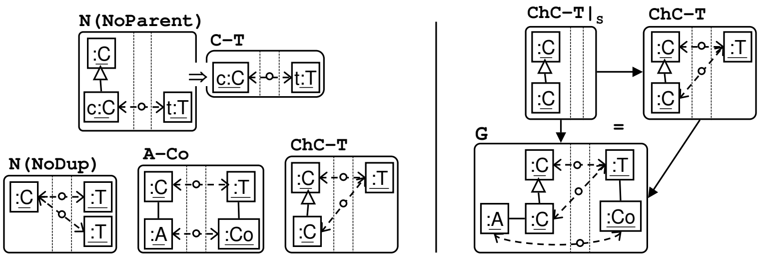


Fig. 1. Example patterns (left). Triple graph satisfying the specification (right).

4 Correctness, Completeness, and Termination of Transformations

An M2M specification S can be used in different scenarios [11]. We can build a target model from a source model (or vice-versa), check whether two models can be mapped according to S, or synchronize two models that previously satisfied S but that were modified separately. Each scenario needs a specialized operational mechanism. Here we cover the first scenario. Starting from a specification S, we generate a triple graph grammar to perform forward transformations. Obviously, the same techniques could be applied for implementing backward transformations. The basic idea is to see the given P-patterns as tiles that have to "cover" a given source model, perhaps with some overlapping. The target model obtained by gluing the target parts of these patterns is the result of the transformation. In addition, the N-patterns allow us to discard some possible models.

Given a source model, a pattern specification will normally have many models. In particular, there may be several non-isomorphic triple graphs sharing the same source graph. This means that there may be several correct transformations for that source graph. Our technique will non-deterministically allow us to obtain all the transformations satisfying the specification. We think that this is the only reasonable approach, if a priori we cannot select any preferred model. It should be obvious that following this kind of approach it is impossible to build some models of the given specification. In particular, it would be impossible to generate models whose target and connection part cannot be generated using the given patterns as described above. For instance, models whose target part includes some nodes of a given type not mentioned in the patterns. We think that restricting our attention to this kind of *generated models* is reasonable in this context. This is similar to the "No Junk" condition in algebraic specification.

Our approach is based on associating to a given specification SP a set of forward transformation rules $TR(SP)$. These rules have, in the left-hand side, a graph including the source part of the conclusion of a positive pattern and part of the target and the connection part. In the right-hand side they have the whole conclusion of the pattern. The idea is that these rules may be used to build "a piece" of the target and the connection graphs, when we discover an occurrence of the source part of a pattern on the given source graph. Rules may include part of the target and connection part of the pattern because this part of the pattern may have been already built by another pattern (rule) application. In addition, the negative premises in the given positive patterns are transformed into NACs of the given rules. Moreover, if we want these rules to be terminating, then we may include some additional NACs that ensure the termination of the set of transformation rules associated to all the P-patterns of a given specification. It should be clear that we can define a set of backward rules in a similar way.

Definition 6 (Forward Transformation Rules for Patterns). *To every P-pattern $\mathcal{S} = \{N(Q \stackrel{n_j}{\to} C_j)\}_{j \in J} \Rightarrow Q$, we associate the set of forward transformation rules $TR(\mathcal{S})$ consisting of all the rules $r = \langle NAC(r), L_r \stackrel{i}{\to} Q \rangle$, where:*

- *L_r is a triple graph such that $Q|_S \subseteq L_r \subset Q$ and i is the monomorphism associated to the inclusion $L_r \subset Q$.*
- *$NAC(r)$ is the set that includes a NAC $n'_j \colon L_r \to C'_j$ for each premise $N(n_j \colon Q \to C_j)$ in $\mathcal{S}$, where n'_j and C'_j are defined up to isomorphism by the pushout depicted on the left below.*

The set of terminating transformation rules associated to $\mathcal{S}$, $TTR(\mathcal{S})$ is the set of all rules $\langle NAC(r) \cup TNAC(r), L_r \stackrel{i}{\to} Q \rangle$ such that $\langle NAC(r), L_r \stackrel{i}{\to} Q \rangle \in TR(\mathcal{S})$ and $TNAC(r)$ is the set of all the termination NACs *for r, i.e. all the monomorphisms $n \colon L_r \to T$ where there is a monomorphism $f_2 \colon Q \to T$ such that n and f_2 are jointly surjective and the diagram on the right below commutes:*

$$
\begin{array}{ccc}
Q|_S & \xrightarrow{(n_j)|_S} & (C_j)|_S \\
{\scriptstyle i^S_{L_r}}\downarrow & po & \downarrow \\
L_r & \xrightarrow[n'_j]{} & C'_j
\end{array}
\qquad\qquad
\begin{array}{ccc}
Q|_S & \xrightarrow{i^S_Q} & Q \\
{\scriptstyle i^S_{L_r}}\downarrow & & \downarrow{\scriptstyle f_2} \\
L_r & \xrightarrow[n]{} & T
\end{array}
$$

Notice that the first condition implies that $Q_S = (L_r)_S$. Notice also that, in the construction of the NAC associated to a premise, $(i^S_{L_r})_S$ is the identity, and so are $(n'_j)_C$ and $(n'_j)_T$.

Example 2. Fig. 2 shows the two forward rules generated from pattern `C-T` presented in Example 1. The first one uses $L = Q|_S$, while the LHS of the second reuses an existing table. Both rules include a NAC (named NAC1), generated from the negative pre-condition `NoParent` of the pattern. The termination NACs ensure that each class is connected to at most one table.

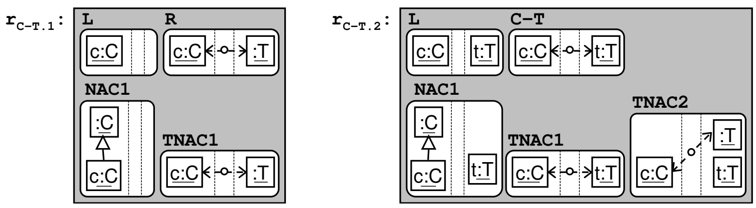


Fig. 2. Forward rules generated from pattern `C-T`.

In some related work (e.g., [6, 4]) termination is ensured by just the termination NAC $L_r \rightarrow Q$. This NAC is enough to ensure finite termination if the set of possible matches of the given rule does not change after applying the transformation rules, i.e. if the possible matches of the rule are always *essential matches*, according to the terminology in [6, 4]. However, this is not the case in our context. Our rules may have triple graphs in the left-hand side with non-empty target or connection part. As a consequence, at some point, there may exist non-essential matches, and this may cause, if we would only use that NAC, that the resulting transformation system is not terminating, as the example below shows.

Example 3. Suppose we have the rule shown to the left of Fig. 3, which is one of the backward rules that we would derive from pattern `ChC-T` if we want to apply our technique to implement backward model transformations. And suppose that we only add a termination NAC (labelled `TNAC`) equal to its RHS. The rule creates a new child of a class connected to a table. Then, the sequence of transformations that starts as shown to the right of the same figure does not terminate. This is so, because the rule adds a new match for the LHS in each derivation, thus being able to produce an inheritance hierarchy of any depth.

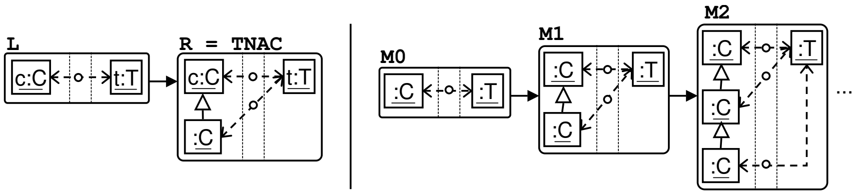


Fig. 3. Backward rule (left). Non-terminating sequence (right).

According to Definition 6, the set of termination NACs for the rule includes the three graphs depicted in Fig. 4. `TNAC2` is isomorphic to `TNAC`, but it identifies the class in L with the child class. Then it is clear that `TNAC2` avoids applying the rule to M_1, thus ensuring termination.

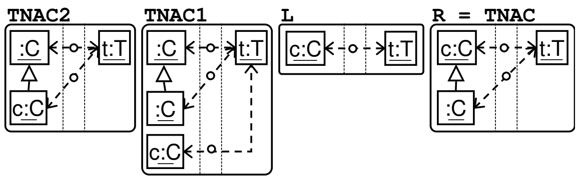


Fig. 4. Rule with termination NACs.

Definition 7 (Forward Transformation Rules for Specifications). *Given a pattern specification SP, we define the set of forward transformation rules associated to SP:*

$$TR(SP) = \bigcup_{\mathcal{S} \in SP^+} TR(\mathcal{S})$$

Similarly, $TTR(SP)$ is the set of terminating transformation rules associated to the patterns in SP^+.

Our first result shows that $TTR(SP)$ is terminating. To show this result, we first need to notice that the transformation rules never modify the source part of the given triple graphs. Then, the key to show this theorem is a specific property of our termination NACs ensuring that, if we have transformed the graph G_1 into the graph G_2 using a rule r with match $m_1 \colon L_r \to G_1$, then we cannot apply the same rule with match $m_2 \colon L_r \to G_2$ if the source parts of the domains of m_1 and m_2 coincide. The reason is that, if we have already applied r with match m_1 then the graph G_2 will already embed L_r and Q (via m_2 and m_1, respectively). Moreover if the source parts of the domains of m_1 and m_2 coincide, then we can ensure the existence of embeddings $n \colon L_r \to T$ and $h \colon T \to G_2$, where n is a termination NAC and $h \circ n = m_2$. Implicitly, this means that the termination

NACs impose finite bounds on the number of times that an element of the given source graph can be part of a match.

Theorem 1 (Termination). *For any finite pattern specification SP, $TTR(SP)$ is terminating.*

Our second main result shows that a triple graph is terminal for $TTR(SP)$ if and only if it forward satisfies the positive patterns in SP. Obviously, we cannot ensure that if G is terminal then G will also satisfy the negative patterns in SP, since they play no role in the construction of $TTR(SP)$.

Theorem 2 (Correctness). *For any finite pattern specification SP, $G \models_F SP^+$ if and only if G is terminal with respect to $TTR(SP)$.*

To prove this theorem, first, we show that a morphism forward satisfies a premise of a pattern if and only if it satisfies the corresponding NACs in the associated rules. Then, we can see that if G is a forward model of SP^+ and h is a match for a rule r associated to a pattern $\mathcal{S}$ that forward satisfies all the negative premises in $\mathcal{S}$, then h will not satisfy a termination NAC in the rule. Conversely, we can prove that if $h\colon Q|_S \to G$ is a monomorphism that satisfies all the premises in $\mathcal{S}$ and h does not satisfy the termination NAC of the rule $Q|_S \to Q$ then there exists an $h'\colon Q \to G$ that extends h.

With respect to completeness, as discussed above, we are only interested in the models whose elements in the target and connection part can be considered to be there because some pattern prescribes that they must be there. We call these graphs SP-generated.

Definition 8 (SP-Generated Graphs). *Given a pattern specification SP, a triple graph G is SP-generated if there is a finite family of P-patterns $\{\mathcal{S}_k\}_{k\in K}$, with $\mathcal{S}_k = \{N(Q_k \xrightarrow{n_j} C_{jk})\}_{j\in J} \Rightarrow Q_k$ in SP, and a family of monomorphisms $\{Q_k \xrightarrow{f_k} G\}_{k\in K}$ such that every f_k forward satisfies all the premises in $\mathcal{S}_k$, and $f_1,\ldots,f_n,i^S_G$ are jointly surjective. In this case, we also say that G is generated by the patterns $\mathcal{S}_1,\ldots\mathcal{S}_n$ and the morphisms $f_1,\ldots f_n$.*

Example 4. Fig. 5 presents three SP-generated graphs from the example specification of Fig. 1, together with the family of patterns that generates them. It must be noted that the same pattern may occur several times in the given family generating the graph. For instance, the pattern `A-Co` is used twice for generating G_3. Graph G_1 is generated by pattern `C-T`, but is not a model of the specification as the child class and its attribute need to be translated. On the contrary, graphs G_2 and G_3 are models of the specification.

Our next result shows that, given a source graph G_S using the rules from $TR(SP)$, and starting from the graph $\langle G_S,\emptyset,\emptyset\rangle$, we can generate exactly all SP-generated graphs H such that $H_S = G_S$. Obviously not all SP-generated graphs need to be models of the specification (for instance $\langle G_S,\emptyset,\emptyset\rangle$ is generated by the empty family of patterns), but this result ensures that if H describes a

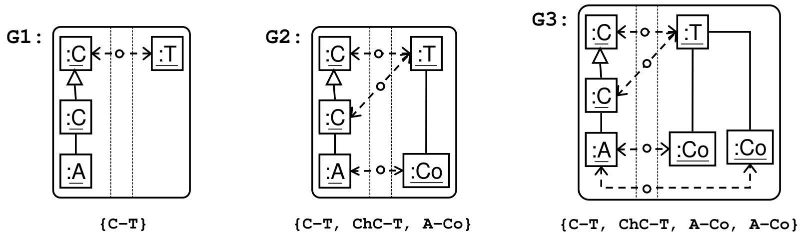


Fig. 5. SP-generated graphs.

valid model transformation of G_S and H only contains nodes and edges that the patterns prescribe that must be present, then H can be obtained by graph transformation.

Theorem 3 (Characterization of SP-Generated Graphs). *Given a pattern specification SP, G is an SP-generated graph if and only if $G|_S \Rightarrow^* G$ using rules from $TR(SP)$.*

The proof of this theorem goes as follows. First, we prove that if G is generated by $\mathcal{S}_1, \ldots \mathcal{S}_n$ and $f_1, \ldots f_n$ then there is a series of transformations:

$$
\begin{array}{ccccccccccc}
L_1 & \xrightarrow{i_1} & Q_1 & & L_2 & \xrightarrow{i_2} & Q_2 & \cdots & L_n & \xrightarrow{i_n} & Q_n \\
\downarrow{\scriptstyle m_1} & & & \searrow{\scriptstyle c_1} & \downarrow{\scriptstyle m_2} & & \downarrow{\scriptstyle c_2} & & \downarrow{\scriptstyle m_n} & & \downarrow{\scriptstyle c_n} \\
G|_S & & \xrightarrow[h_1]{} & & G_1 & \xrightarrow[h_2]{} & G_2 & \xrightarrow[h_3]{} \cdots \xrightarrow[h_{n-1}]{} & G_{n-1} & \xrightarrow[h_n]{} & G_n
\end{array}
$$

such that $G = G_n$ up to isomorphism, where the rules involved in these transformations are obtained by the pullback:

$$
\begin{array}{ccc}
L_k & \xrightarrow{m_k} & Q_k \\
\downarrow{\scriptstyle i_k} & & \downarrow{\scriptstyle f_k} \\
G_{k-1} & \xrightarrow[h_n \circ \cdots \circ h_k]{} & G
\end{array}
$$

Conversely, we can prove that if G can be obtained by a series of transformations as the one above then G is generated by the patterns associated to these rules, together with the morphisms $f_1, \ldots f_n$, where $f_k = h_n \circ \cdots \circ h_{k+1} \circ c_k$.

As a direct consequence of this theorem we immediately get our first completeness result:

Corollary 1 (Completeness). *Given a pattern specification SP, if G is SP-generated and $G \models_F SP$ then $G|_S \Rightarrow^* G$ using rules from $TR(SP)$.*

There are two aspects in the previous completeness result which may be considered not fully satisfactory. On one hand, we have proved completeness of $TR(SP)$, i.e. a non-terminating transformation system. On the other hand, the notion of generated model may not completely follow our intuition. In particular, according to Def. 8, a given pattern may be used several times with the same match to generate several different parts of the target and connection graphs. Next, we provide a more restrictive notion of SP-generated graphs, namely strictly SP-generated graphs, and then we show that strictly SP-generated forward models are the terminal graphs of our terminating transformation systems.

Definition 9 (Strictly SP-Generated Graphs). *Given a pattern specification SP, a triple graph G is strictly SP-generated if G is an SP-generated graph and for every P-pattern $\mathcal{S} = \{N(Q \stackrel{n_j}{\to} C_j)\}_{j\in J} \Rightarrow Q$, if $f_1 \colon Q \to G$ and $f_2 \colon Q \to G$ are two monomorphisms such that $(f_1)_S = (f_2)_S$ and both forward satisfy all the premises $n_j \colon Q \to C_j$, then $f_1 = f_2$.*

Notice that in the above definition it is enough to ask that either f_1 or f_2 forward satisfy all the premises of the pattern since this depends only on the source component of the morphisms. Therefore, since both morphisms coincide in their source component, if one of them forward satisfies a premise so will do the other morphism.

Example 5. In Fig. 5, graphs G_1 and G_2 are strictly generated, while G_3 is not, because both occurrences of `A-Co` share the same source.

Theorem 4 (Completeness for strictly SP-Generated Graphs). *Given a pattern specification SP, if G is strictly SP-generated and $G \models_F SP$ then $G|_S \Rightarrow^* G$ using rules from $TTR(SP)$ and, moreover, G is a terminal graph.*

The key to prove the above theorem is to show that if G is a strictly SP-generated graph and we assume that G is generated by a minimal family of patterns $\mathcal{S}_1, \dots \mathcal{S}_n$ and monomorphisms $f_1, \dots f_n$, then the minimality of the family ensures that all the matches in the above derivation satisfy the corresponding termination NACs, which means that the rules may be applied.

Finally, by Theorem 2 and Theorem 4, we have:

Corollary 2 (Soundness and Completeness). *Given a pattern specification SP consisting of positive patterns, and a strictly SP-generated graph G then $G \models_F SP$ if and only if $G|_S \Rightarrow^* G$ using rules from $TTR(SP)$ and G is a terminal graph.*

Remark 1. Corollary 2 tells us that, given a set of positive patterns SP and a source graph G_S, the set of all strictly SP-generated forward models of SP, whose S-component coincides with G_S, is included in the set of terminal graphs obtained from $\langle G_S, \emptyset, \emptyset\rangle$. However, if SP includes some negative patterns, then some (or perhaps all) of these terminal graphs may fail to satisfy these additional patterns. As said above, this is completely reasonable since negative patterns have not played any role in the construction of $TR(SP)$ or $TTR(SP)$. However, the negative patterns can be added as NACs into the transformation rules as described, for instance, in [6]. In this case, it will be impossible to transform a graph G_1 into G_2 if G_2 would violate some negative pattern. Then, the transformation system could be considered more efficient since the derivation tree associated to a given start graph would be pruned from all the graphs violating the negative patterns. However, in this case, our soundness and completeness results would slightly change. In particular, a terminal graph would not necessarily be a model of the given positive patterns anymore. More precisely, a graph would be terminal if it is a model of the given positive patterns or if all its possible transformations violate a negative pattern.

Example 6. Fig. 6 shows some derivations starting from a given graph G_0 using the generated terminating forward rules. All graphs in the derivations are strictly SP-generated. Hence all graphs in Fig. 5 are reachable. Notice that G_3 in Fig. 5 is not reachable using the terminating rules, as the rule generated from `A-Co` is not applicable to G_2. Graphs G_2 and G_5 are terminal w.r.t. $TTR(SP)$: the former is a forward model of SP, and the latter is only a forward model of SP^+ because the N-pattern `N(NoDup)` is not satisfied. As stated in the remark, we could add additional NACs to the generated rules to forbid applying a rule creating an occurrence of N-patterns. In that case, rule $r_{A-Co.1}$ would not be applied to G_4 and therefore graph G_5 would not be reached.

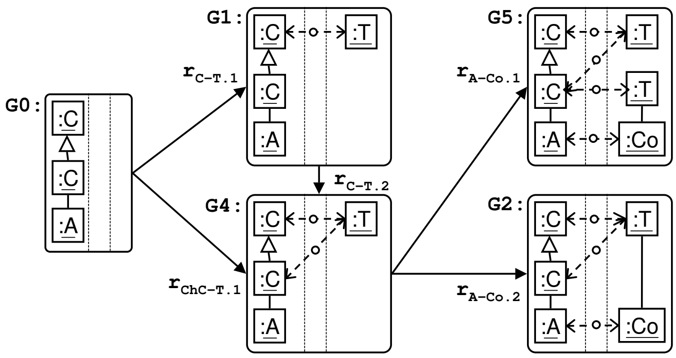


Fig. 6. Some derivations using the generated terminating forward rules.

5 Related Work

Some declarative approaches to M2M transformation are unidirectional, e.g. PMT [19] or Tefkat [13], while we generate both forward and backward transformations. Among the visual, declarative and bidirectional approaches, a prominent example is the OMG's standard language QVT-relational [16]. Relations in QVT contain *when* and *where* clauses to guide the execution of the operational mechanisms. In our case, it is not necessary because the generated rules are terminating, correct and complete. Moreover, QVT lacks a formal semantics, complicating the analysis. On the contrary, our patterns have a formal semantics, which makes them amenable to verification.

TGGs [17] formalize the synchronized evolution of two graphs through declarative rules. From this specification, low level operational TGG rules are derived to perform forward and backward transformations, similar to our case. However, whereas in declarative TGG rules dependencies must be made explicit (i.e. TGG rules must declare which elements should exist and which ones are created), in our patterns this information is derived when generating the rules.

Completeness and correctness of forward and backward transformations was proved for TGGs in [7]. Termination was not studied because TGGs need a control mechanism to guide the execution of the operational rules, such as priorities [10] or their coupling to editing rules [5]. This is not necessary with our patterns, but we need to ensure finite termination of the operational mechanisms. The conditions for information preservation of forward and backward transformations was studied in [5]. Moreover, in [8] the results in [5] are extended to the case of triple graph grammars with NACs. A similar result can be adapted for pattern-based specifications.

In our initial presentation of pattern-based transformation [3], we introduced some deduction operations able to generate new patterns from existing ones. In the present paper, we have simplified the framework by eliminating such deduction operations, but enriching the process of generating operational rules. The new generation process ensures completeness as it generates each possible LHS for the rules, which however could not be guaranteed with the deduction operations. However, such operations can be used as heuristics to generate less rules, or to reduce the non-confluent behaviour of the transformations.

6 Conclusions and Future Work

In this paper we have demonstrated three properties of the compilation mechanisms of pattern-based M2M specifications into graph grammar rules: finite termination, correctness with respect to the positive fragment of the specification and completeness. The first result allows using the generated rules without any external control mechanism for rule execution, as a difference from current approaches [11, 16, 17]. The correctness result ensures soundness of the operational mechanisms, and as remarked in the article, it can be easily extended to correctness of specifications including negative patterns. Finally, completeness guarantees that if the M2M specification has a model, then it can be found by the operational mechanism.

Our results provide a formal foundation for our approach to pattern-based M2M transformations. However, from a practical point of view, we believe that the techniques presented in this paper have to be complemented with other techniques that ensure some good performance. More precisely, our results guarantee that using our approach we can obtain all the (generated) models of given specification, which means all the possible transformations that are correct according to the given specification. However, on one hand, this is in general an exponential number of models, which implies that computing all these models would be not feasible. On the other hand, typically, there may be some preferred kind of model (for instance, models that are minimal in some sense) and, as a consequence, we would not be interested in computing all the models, but only the preferred ones. In addition, in order to make pattern-based transformation useful for Model-Driven Development, we are currently addressing further challenges: handling attributes in M2M specifications, supporting advanced meta-modelling concepts like inheritance and integrity constraints, and tool support. Moreover, we believe that pattern-based transformation can be used as a formal basis for other transformation languages, like QVT.

Acknowledgments. Work supported by the Spanish Ministry of Science and Innovation, projects METEORIC (TIN2008-02081), MODUWEB (TIN2006-09678) and FORMALISM (TIN2007-66523). Moreover, part of this work was done during a sabbatical leave of the first author at TU Berlin, with financial support from the Spanish Ministry of Science and Innovation (grant ref. PR2008-0185). We thank the referees for their useful comments.

References

1. D. H. Akehurst and S. Kent. A relational approach to defining transformations in a metamodel. In *Proc. UML'02*, volume 2460 of *LNCS*, pages 243–258. Springer, 2002.
2. P. Braun and F. Marschall. Transforming object oriented models with BOTL. *ENTCS*, 72(3), 2003.
3. J. de Lara and E. Guerra. Pattern-based model-to-model transformation. In *Proc. ICGT'08*, volume 5214 of *LNCS*, pages 426–441. Springer, 2008.
4. H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, and S. Varró-Gyapay. Termination criteria for model transformation. In M. Cerioli, editor, *FASE*, volume 3442 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2005.
5. H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. In *Proc. FASE'07*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.
6. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer-Verlag, 2006.
7. H. Ehrig, C. Ermel, and F. Hermann. On the relationship of model transformations based on triple and plain graph grammars. In *Proc. GRAMOT'08*, 2008.
8. H. Ehrig, F. Hermann, and C. Sartorius. Completeness and correctness of model transformations based on triple graph grammars with negative application conditions. In *GT-VMT 2009*, Electronic Communications of the EASST, to appear, 2009.
9. R. Heckel and A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. *ENTCS*, 2, 1995.
10. A. Königs. Model transformation with triple graph grammars. In *Proc. MTiP'05*, 2005.

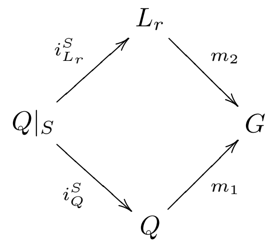
11. A. Königs and A. Schürr. Tool integration with triple graph grammars - a survey. *ENTCS*, 148(1):113–150, 2006.
12. S. Lack and P. Sobocinski. Adhesive categories. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2004.
13. M. Lawley and J. Steel. Practical declarative model transformation with Tefkat. In *MoDELS Satellite Events*, volume 3844 of *LNCS*, pages 139–150. Springer, 2005.
14. MOLA. MOdel transformation LAnguage. `http://mola.mii.lu.lv/`.
15. MTF. Model Transformation Framework. `http://www.alphaworks.ibm.com/tech/mtf`.
16. QVT. `http://www.omg.org/docs/ptc/05-11-01.pdf`, 2005.
17. A. Schürr. Specification of graph translators with triple graph grammars. In *Proc. WG'94*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.
18. P. Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. In *Proc. MoDELS'07*, volume 4735 of *LNCS*, pages 1–15. Springer, 2007.
19. L. Tratt. A change propagating model transformation language. *JOT*, 7(3):107–126, 2008.

Appendix: Proofs of the results

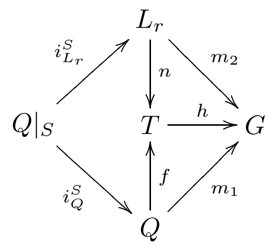
The proposition below shows a property of our termination NACs which is the key to prove the finite termination of the set of transformation rules associated to a given specification. In particular, the termination NACs ensure that, if we have transformed the graph G_1 into the graph G_2 using a rule r with match $m : L_r \to G_1$, i.e. if there is an injective morphism $m_1 : Q \to G_2$ such that m_1 coincides with m when restricted to L_r, then we cannot apply the same rule with match $m_2 : L_r \to G_2$ if $(m_1)_S = (m_2)_S$:

Proposition 2. *Given a rule* $\langle NAC(r) \cup TNAC(r), L_r \xrightarrow{i} Q\rangle$ *associated to a positive pattern and given a graph* G *such that there is an injective morphism* $m_1 : Q \to G$*. If* $m_2 : L_r \to G$ *is a monomorphism such that* $(m_1)_S = (m_2)_S$ *then there is a termination NAC* $(L_r \xrightarrow{n} T) \in TNAC(r)$ *and a monomorphism* $h : T \to G$ *such that* $h \circ n = m_2$*.*

Proof. We know that $m_1 \circ i^S_Q = \langle (m_1)_S, (1_G)_C, (1_G)_T\rangle$, where 1_G is the canonical morphism from the empty graph to G. But, since $Q_S = (L_r)_S$ and by the assumption that $(m_1)_S = (m_2)_S$, we have that $(m_1)_S \circ (i^S_Q)_S = (m_2)_S \circ (i^S_{L_r})_S$. This means that the diagram below commutes:



Then, by the n-factorization property (Prop. 1) there exists a graph T and monomorphisms h, f and n:



such that f and n are jointly surjective and the diagram above commutes. ■

Proof of Theorem 1 Let G_0 be an arbitrary triple graph and let us suppose that there exists an infinite sequence of transformations:

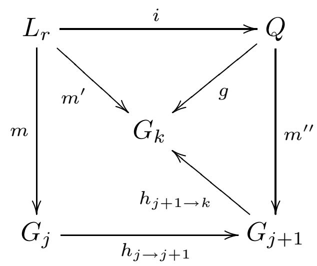
$$G_0 \Rightarrow_{TTR(SP)} G_1 \Rightarrow_{TTR(SP)} \cdots \Rightarrow_{TTR(SP)} G_k \Rightarrow_{TTR(SP)} \cdots$$

This means that there are monomorphisms:

$$G_0 \to G_1 \to \cdots \to G_k \to \dots$$

Let us denote by $h_{j \to k}$ the monomorphism from G_j to G_k induced by the transformations. Now, since there is a finite set of rules, there should exist a rule $r \in TTR(SP)$, $r = \langle NAC(r) \cup TNAC(r), L_r \xrightarrow{i} Q \rangle$, that is applied infinitely often in the sequence. In addition, we know that the transformation rules do not modify the source part, i.e. for every j, k, $(h_{j \to k})_S$ is the identity. But, since the source part of G_0 is a finite graph there should exist two matches $m : L_r \to G_j$ and $m' : L_r \to G_k$, with $j < k$, such that $m_S = m'_S$.

If we have applied the rule r with match m at step j, we have that G_{j+1} is obtained by the pushout depicted in the outer square:

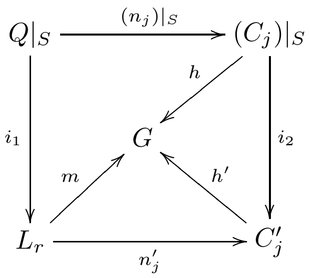


This means that $g = h_{j+1 \to k} \circ m''$ is a monomorphism from Q to G_k such that $g_S = m'_S$. Then, by Prop. 2, r includes a termination NAC $n : L_r \to T$ such that there is a monomorphism $h : T \to G_k$ with $h \circ n = m'$. But this means that this NAC would not allow the application of r with match m' against the hypothesis. ■

Before proving our soundness and completeness results, we will prove a technical proposition, that shows that a morphism forward satisfies a precondition of a pattern if and only if it satisfies its corresponding NAC in the associated rules.

Proposition 3. *Let $\langle NAC(r) \cup TNAC(r), L_r \xrightarrow{i} Q \rangle$ be a rule associated to the pattern $\mathcal{S} = N(\{Q \xrightarrow{n_j} C_j\}_{j \in J}) \Rightarrow Q$, where the NAC $n'_j : L_r \to C'_j$ corresponds to the precondition $n_j : Q \to C_j$, and let $m : L_r \to G$ be a monomorphism. Then there is a monomorphism $h : (C_j)|_S \to G$ such that $h \circ (n_j)|_S = m \circ i_1$ if and only if there is a monomorphism $h' : C'_j \to G$ such that $h' \circ n'_j = m$, where i_1 is the monomorphism associated to the inclusion $Q|_S \subseteq L_r$.*

Proof. Suppose that there is a monomorphism $h : (C_j)|_S \to G$ such that $h \circ (n_j)|_S = m \circ i_1$. By the universal property of pushouts there exists a morphism h' making the diagram below commute:



Thus, we have to prove that h' is injective. But this is not difficult. It is enough to consider each of the components of h'. In particular, by construction, we know that $C'_j = \langle (C_j)_S, (L_r)_C, (L_r)_T \rangle$, which means that $(i_2)_S = id$, $n'_C = id$ and $n'_T = id$. As a consequence, $h' = \langle h_S, m_C, m_T \rangle$. But, since m and h are assumed to be injective, so should be h'.

Conversely, suppose that $h' : C'_j \to G$ is a monomorphism such that $h' \circ n'_j = m$. Then, if we define $h = h' \circ i_2$ (see the diagram above), we have that $h \circ (n_j)|_S = h' \circ i_2 \circ (n_j)|_S = h' \circ n'_j \circ i_1 = m \circ i_1$. ■

Proof of Theorem 2

- Let us suppose that $G \models_F SP^+$ this means that for every pattern $\mathcal{S} = N(\{Q \stackrel{n_j}{\to} C_j\}_{j \in J}) \Rightarrow Q$ in SP^+, $G \models_F \mathcal{S}$ and this means that for every monomorphism $m : Q|_S \to G$, such that for every $j \in J$, $m \models_F Q \stackrel{n_j}{\to} C_j$, there exist a monomorphism $m' : Q \to G$ such that $m = m' \circ i^S_Q$.

  Let $r = \langle NAC(r) \cup TNAC(r), L_r \stackrel{i}{\to} Q \rangle$ be a rule in $TTR(\mathcal{S})$, we will show that if $h : L_r \to G$ is a match for the rule then there is some NAC in $NAC(r)$ that forbids the application of r.

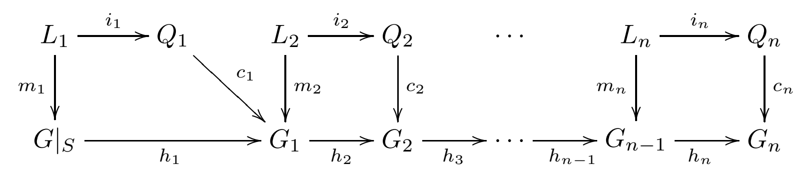
  If $h : L_r \to G$ is a monomorphism then so it is $m : Q|_S \to G$, where $m = h \circ i_1$ and i_1 is the monomorphism associated to the inclusion $Q|_S \subseteq L_r$.Then, since $G \models_F \mathcal{S}$, we have two cases:

  1. m does not forward satisfy some negative precondition. This means, by Prop. 3, that h does not satisfy its corresponding NAC and, as a consequence, the rule r would not be applicable with match h.
  2. m satisfies all the preconditions and there is a monomorphism $m' : Q \to G$ such that $m' \circ i^S_Q = m = h \circ i_1$. But $(i_1)_S$ and $(i^S_Q)_S$ are the identity and this means that $m'_S = h_S$. Then, by proposition 2 there is a termination NAC in r such that h does not satisfy that NAC.

- Conversely, suppose that G is a terminal graph with respect to $TTR(SP)$, let us see that G satisfies all the positive patterns in SP. Let $\mathcal{S} = N(\{Q \stackrel{n_j}{\to} C_j\}_{j \in J}) \Rightarrow Q$ be a pattern in SP^+, we have to show that if $m : Q|_S \to G$ is a monomorphism, and for every $j \in J$, $m \models_F Q \stackrel{n_j}{\to} C_j$, there exist a monomorphism $m' : Q \to G$ such that $m = m' \circ i^S_Q$. Let r be the terminating rule $r = \langle NAC(r) \cup TNAC(r), Q|_S \to Q \rangle$ associated to that pattern. Notice that in this case $L_r = Q|_S$. Hence, by construction, the NAC associated to the precondition $n_j : Q \to C_j$ is the monomorphism $(n_j)|_S : Q|_S \to (C_j)|_S$ and the only termination NAC is the inclusion $i : Q|_S \to Q$. Now, if $h : Q|_S \to G$ is a monomorphism then h is a possible match for the rule r. Moreover, if h satisfies the precondition $n_j : Q \to C_j$ then h satisfies the NAC $(n_j)|_S : Q|_S \to (C_j)|_S$. Then, if G is a terminal graph with respect to $TR(SP)$, this may only mean that G does not satisfy the termination NAC $i : Q|_S \to Q$. But this means that there exist a monomorphism $f : Q \to G$ such that $f \circ i = h$. ■

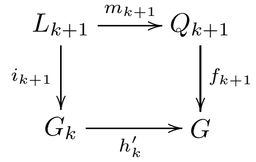
Proof of Theorem 3

We start proving that if G is an SP-generated graph then $G|_S \Rightarrow^*_{TR(SP)} G$. Let us assume that $\mathcal{S}_1, \ldots \mathcal{S}_n,\ f_1, \ldots f_n$ are the patterns and morphisms that generate G. We will show that there is a series of transformations:



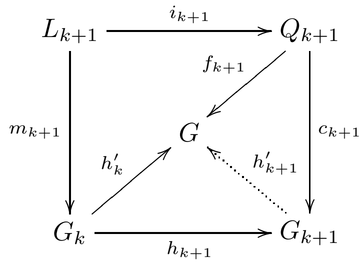
such that $G = G_n$ up to isomorphism. More precisely, first we will show by induction that for every k, if $\mathcal{S}_k = N(\{Q_k \stackrel{n_j}{\to} C_{jk}\}_{j \in J}) \Rightarrow Q_k$, then there is a rule $r_k = \langle NAC(r), L_k \stackrel{i_k}{\to} Q_k\rangle$ in $TR(\mathcal{S}_k)$ and there is an injective morphism $h'_k : G_k \to G$ such that $(h'_k)_S$ is the identity (up to isomorphism), $f_k = h'_k \circ c_k$ and $h'_k = h'_{k+1} \circ h_{k+1}$. Notice that the last condition implies that $h'_k = h'_n \circ h_n \circ \cdots \circ h_{k+1}$. Finally, we will show that h'_n is an isomorphism.

- If $n = 0$ then trivially the theorem holds, since this means that $G = G|_S$.
- Let us suppose that $G|_S \Rightarrow^* G_k$ and $h'_k : G_k \to G$ is a monomorphism such that $(h'_k)_S$ is an isomorphism. Let r_{k+1} be the rule in $TR(SP)$ whose right-hand side is Q_{k+1} and whose left-hand side, $L_{k+1} \subseteq Q_{k+1}$, is defined (up to isomorphism) by the following pullback, with inclusion i_{k+1} without loss of generality:



Notice that there is such rule in $TR(\mathcal{S}_{k+1})$, since we know that f_{k+1} is a monomorphism from Q_{k+1} to G and we also know that $(h'_k)_S$ is the identity, which means that there is a monomorphism from $(Q_{k+1})|_S$ to G, defined by $(f_{k+1})_S$, and therefore, by the pullback property of L_{k+1}, $(Q_{k+1})|_S \subseteq L_{k+1}$.

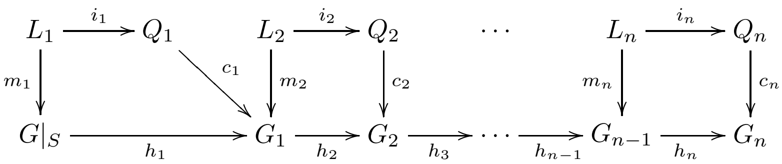
Now, to apply this rule to G_k with match m_{k+1} we must first show that m_{k+1} satisfies the NACs in $NAC(r_{k+1})$. But this is a consequence of Prop. 3. Therefore, we know that we can apply the rule r_{k+1} with match m_{k+1} to G_k yielding the graph G_{k+1} and, by the universal property of pushouts, we know that there is a morphism $h'_{k+1} : G_{k+1} \to G$ such that $f_{k+1} = h'_{k+1} \circ c_{k+1}$ and $h'_k = h'_{k+1} \circ h_{k+1}$:



Now, since the outer square is a pushout, the inner diagram is a pullback, the morphisms in the diagram are monomorphisms, and triple graphs are an adhesive category, then h'_{k+1} is a monomorphism. Moreover, since $h'_k = h'_{k+1} \circ h_{k+1}$, $(h'_{k+1})_S$ is a monomorphism and $(h'_k)_S$ and $(h_{k+1})_S$ are isomorphisms we can conclude that $(h'_{k+1})_S$ is an isomorphism.

To end this part of the proof we must show that h'_n is an isomorphism. We know that h'_n is a monomorphism and $(h'_n)_S$ is an isomorphism, so we have to prove that $(h'_n)_T$ and $(h'_n)_C$ are also surjective. Let e be an element in G_T (a node or an edge). Since the morphisms $f_1, \ldots, f_n, i^S_G$ are jointly surjective and $(G|s)_T$ is the empty graph then $(f_1)_T, \ldots, (f_n)_T$ are jointly surjective. As a consequence, there exists an element $e_k \in (Q_k)_T$ such that $e = f_k(e_k)$. But, since $f_k = h'_k \circ c_k = h'_n \circ h_n \circ \cdots \circ h_{k+1} \circ c_k$ this means that there is an element $e_n \in G_n$, $e_n = h_n \circ \cdots \circ h_{k+1} \circ c_k(e_k)$ such that $h'_n(e_n) = f_k(e_k) = e$. Similarly, we could prove that $(h_n)_C$ is surjective.

Let us now prove that if $G|_S \Rightarrow^*_{TR(SP)} G$ then G is SP-generated. So, suppose that we have a derivation:

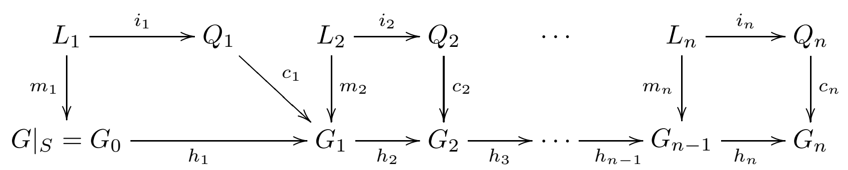


with $G_n = G$. We will prove that, if we define the morphisms $f_1 = h_n \circ \cdots \circ h_2 \circ c_1, \ldots f_k = h_n \circ \cdots \circ h_{k+1} \circ c_k, \ldots, f_n = c_n$, then $f_1, \ldots, f_n, i^S_G$ are jointly surjective. Now, since G is the result of the sequence of pushouts of the above diagram, this means that G is the colimit of that diagram, and this implies that the morphisms $f_1, \ldots, f_n, h_n \circ \cdots \circ h_1$ are jointly surjective. But, $(h_n \circ \cdots \circ h_1)_S$ is the equality since the rules in $TR(SP)$ do not modify the source part of the triple graphs. As a consequence, $h_n \circ \cdots \circ h_1 = i^S_G$, which means that $f_1, \ldots, f_n, i^S_G$ are jointly surjective. ■

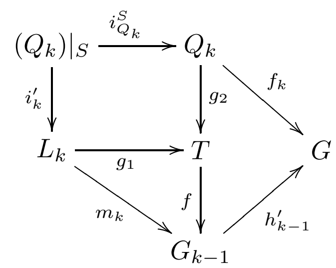
Proof of Theorem 4

Let G be a strictly SP-generated forward model of SP. Since G is an SP-generated graph, we know that G is generated by some finite families of patterns $\{\mathcal{S}_k\}_{k\in K}$ and morphisms $\{f_k\}_{k\in K}$. Let $\{\mathcal{S}_k\}_{k\in K}$ and $\{f_k\}_{k\in K}$ be minimal in the sense that, for any $K' \subset K$, G is not generated by $\{\mathcal{S}_k\}_{k\in K'}$ and $\{f_k\}_{k\in K'}$.

As we have seen in the proof of Theorem 3, if $\{\mathcal{S}_k\}_{k\in K}$ consists of the patterns $\mathcal{S}_1, \ldots \mathcal{S}_n$, and $\{f_k\}_{k\in K}$ consists of the morphisms $f_1, \ldots f_n$, then there is a $TR(SP)$-derivation:



with $G_n = G$, where each rule r_k is associated to the pattern $\mathcal{S}_k$ and where $f_k = h'_k \circ c_k$, where $h'_k = h_n \circ \cdots \circ h_{k+1} : G_k \to G$. So we will prove that we can produce the same derivation applying the corresponding rules in $TTR(SP)$. More precisely, we need to prove that, for every k, m_k satisfies all the termination NACs in the rule $\langle NAC(r_k) \cup TNAC(r_k), L_k \xrightarrow{i_k} Q_k\rangle$, which are all the monomorphisms $g_1 : L_k \to T$ where there is a monomorphism $g_2 : Q_k \to T$ such that g_1 and g_2 are jointly surjective and the square subdiagram on the top-left corner of the diagram below commutes:



and where i'_k is the inclusion $(Q_k)|_S \subseteq L_k$. Let us suppose that there is a monomorphism $f : T \to G_{k-1}$ such that $f \circ g_1 = m_k$. Then we have that $h'_{k-1} \circ f \circ g_1 \circ i'_k = h'_{k-1} \circ m_k \circ i'_k = h'_k \circ h_k \circ m_k \circ i'_k = h'_k \circ c_k \circ i_k \circ i'_k = f_k \circ i_k \circ i'_k$, using $h'_k \circ c_k = f_k$. On the other hand we have that $h'_{k-1} \circ f \circ g_1 \circ i'_k = h'_{k-1} \circ f \circ g_2 \circ i^S_{Q_k}$, which means $h'_{k-1} \circ f \circ g_2 \circ i^S_{Q_k} = f_k \circ i_k \circ i'_k$ and, hence, $(h'_{k-1} \circ f \circ g_2)_S = (f_k)_S$.

Therefore, we have two monomorphisms f_k and $h'_{k-1} \circ f \circ g_2$ from Q_k to G such that their source component coincides. Moreover, we know that f_k satisfies all the preconditions in the pattern $\mathcal{S}_k$, which means that $h'_{k-1} \circ f \circ g_2$ also satisfies all the preconditions in $\mathcal{S}_k$. Hence, since G is assumed to be strictly

SP-generated, $f_k = h'_{k-1} \circ f \circ g_2$. But this means that G is generated by the patterns $\{\mathcal{S}_1, \dots \mathcal{S}_n\} \setminus \{\mathcal{S}_k\}$, and the morphisms $\{f_1, \dots f_n\} \setminus \{f_k\}$, since every element in the image of f_k is also on the image of h'_k, which means that this element is in the image of $f_1, \dots f_{k-1}$. Therefore $\mathcal{S}_1, \dots \mathcal{S}_n$ and $f_1, \dots f_n$ would not be minimal, against our original assumption.

Finally, by Theorem 2, we know that if G_n is a model of SP, then G_n is terminal with respect to $TTR(SP)$. ■