The Formal Specification Column

by

Hartmut Ehrig

Technical University of Berlin, Department of Computer Science
Franklinstraße 28/29, D-10587 Berlin, Germany
ehrig@cs.tu-berlin.de

Model Transformations by Graph Transformation are Functors

Hartmut Ehrig[1], Karsten Ehrig[2], Claudia Ermel[1], Ulrike Prange[1]

[1]Fac. of Electr. Engineering and Comp. Science
Technical University of Berlin, Germany
ehrig|lieske|uprange@cs.tu-berlin.de

[2] Department of Computer Science
University of Leicester, United Kingdom
karsten@mcs.le.ac.uk

Abstract

In this paper, we extend our ideas on model transformations as functors discussed in the previous issue and embed this concept into the framework of graph transformation systems. We show that under certain restrictions of the rules model transformations by graph transformation are functors.

Introduction

In our previous column [2] we have discussed the claim that "model transformations should be functors" from a mathematical and from a practical point of view. Especially interesting from the practical side was the proposal of the POPL'07

Bulletin of the EATCS no 93, pp. 134–142, October 2007

keynote speaker Don Batory [1] that model transformations in his approach of Feature Oriented Model Driven Development (FOMDD) should be functors.

On the theoretical side, we have discussed well-known data type constructions which are functors and can be considered as model transformations. Typical examples are the semantics of parameterized specification and module specification (see [4, 5]). Moreover we have claimed that under suitable conditions model transformations defined by graph transformations are functors. In this column, we justify this last claim.

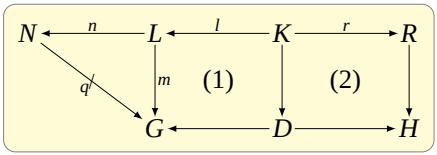
1 Model Transformation by Graph Transformation

In this section, we define model transformation by graph transformation. First we shortly introduce the necessary definitions of typed graphs, rules and transformations (see [3]).

A graph $G = (V, E, src, tar)$ is given by sets V and E of nodes and edges, respectively, and source and target functions $src, tar : E \to V$. For graphs $G^i = (V^i, E^i, src^i, tar^i)$ with $i = 1, 2$, a graph morphism $f = (f_V, f_E) : G^1 \to G^2$ is given by mappings $f_V : V^1 \to V^2$ and $f_E : E^1 \to E^2$ compatible with the source and target functions, i.e. $f_V \circ src^1 = src^2 \circ f_E$ and $f_V \circ tar^1 = tar^2 \circ f_E$. Graphs and graph morphisms form the category **Graphs**.

A type graph TG is a distinguished graph that defines node and edge types. A typed graph (G, t) is a graph G together with a typing morphism $t : G \to TG$. If the typing is clear in the context, we denote G as a typed graph without explicitly naming t. A typed graph morphism f between typed graphs (G^1, t^1) and (G^2, t^2) is a graph morphism $f : G^1 \to G^2$ compatible with the typing, i.e. $t^2 \circ f = t^1$. For a type graph TG, typed graphs and typed graph morphisms form the category $\mathbf{Graphs_{TG}}$.

A typed graph rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consists of three typed graphs L, K and R, called left hand side, gluing graph and right hand side, respectively, and injective typed graph morphisms l and r. Given a rule p and a typed graph morphism $m : L \to G$, called match, the application of the rule p to G via the match m is given by the following two pushouts (1) and (2) leading to the direct typed graph transformation $G \overset{p,m}{\Longrightarrow} H$. A sequence $G_0 \Longrightarrow G_1 \Longrightarrow \ldots \Longrightarrow G_n$ of direct transformations is then called transformation and denoted by $G_0 \overset{*}{\Longrightarrow} G_n$.



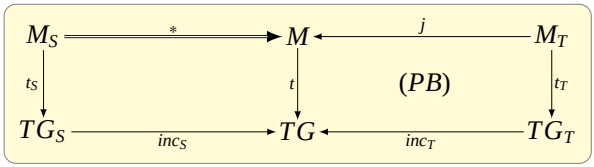
A negative application condition (NAC) for p is a typed graph morphism $n : L \to N$. The match m satisfies the NAC n if there does not exist an injective morphism $q : N \to G$ such that $q \circ n = m$. For p we define a set NAC_p of NACs. The application of p with NACs is allowed only if m satisfies all NACs $n \in NAC_p$.

A graph transformation system $GTS = (TG, Prod)$ is now given by a type graph TG and a set $Prod$ of rules with NACs. For graph transformation systems there are many interesting results like the Local Church–Rosser, Parallelism, Embedding, Extension and Local Confluence Theorems [3].

In the following, we only consider nondeleting rules. In this case we have $L = K$ and the rule is completely defined by the morphism r and its NACs.

For the definition of a model transformation by graph transformation, the source and target models have to be given as typed graphs typed over TG_S and TG_T, respectively, where TG_S is the type graph defining the source language L_S and TG_T is the type graph defining the target language L_T. Performing model transformations by typed graph transformations means taking a model as a typed graph and transforming it according to certain rules. The result is a typed graph which represents the target model.

For the model transformation, we define a type graph TG with $TG_S \subseteq TG$ and $TG_T \subseteq TG$. This type graph includes not only TG_S and TG_T, but may contain additional node and edge types which are needed during the transformation process. Due to the inclusion, all source and target models are also automatically typed over TG. Given a graph transformation system $GTS = (TG, Prod)$, for a source model M_S we start the model transformation by applying the rules in $Prod$ as long as possible. If this process terminates, it results in a transformation $M_S \overset{*}{\Longrightarrow} M$ with a graph M typed over TG. M contains the target model, but also the source model and possibly additional nodes and edges. To obtain the target model, we restict M to the target type graph TG_T by constructing the pullback (PB). The pullback object $M|_{TG_T} = M_T$ is our target model and correctly typed over TG_T.



Given the graph transformation system $GTS = (TG, Prod)$, we define the model transformation $MT : \mathbf{L_S} \Rightarrow \mathbf{L_T}$, where $\mathbf{L_S} \subseteq \mathbf{Graphs_{TG_S}}$ and $\mathbf{L_T} \subseteq \mathbf{Graphs_{TG_T}}$ are subcategories with $M_S \in \mathbf{L_S}$ and $M_T \in \mathbf{L_T}$ if and only if $M_S \overset{*}{\Longrightarrow} M$ is a terminating transformation with $M|_{TG_T} = M_T$, and all injective morphisms.

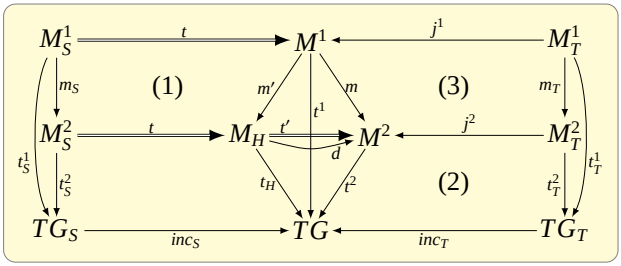
In general, there may be different terminating transformations leading to target models M_T^1 and M_T^2 for the same source models M_S. If $MT(M_S)$ is uniquely

defined, we say MT has functional behaviour.

2 Model Transformation as a Functor

In this section, we consider a model transformation $MT : \mathbf{L_S} \Rightarrow \mathbf{L_T}$ with functional behaviour given by a graph transformation system $GTS = (TG, Prod)$ as described in Section 1. In addition, we restrict the rules in $Prod$ to $TG \backslash TG_S$-generating rules, where for a rule $p : L \xrightarrow{r} R$ there are only elements $x \in R \backslash r(L)$ with type $t^R(x) \in TG \backslash TG_S$, and define that $NAC_p = \{r\}$, i.e. each rule $p : L \xrightarrow{r} R$ has only one NAC, which is exactly the right hand side. With these restrictions, we want to show that MT becomes a functor.

First, we have to define $MT(m_S)$ for an injective morphism $m_S : M_S^1 \to M_S^2$ in $\mathbf{L_S}$. For M_S^1, we have a transformation sequence $M_S^1 \overset{t}{\Longrightarrow} M^1$ and a restriction M_T^1 leading to the model transformation $MT(M_S^1) = M_T^1$. Since only nondeleting rules are applied in t, m_S is boundary-consistent w.r.t. t [6], i.e. the transformation sequence does not delete any boundary element of n. Moreover, due to the restrictions of the NACs and m_S being injective, m_S is also NAC-consistent w.r.t. t [6], which means that no NAC of t is violated by extending M_S^1 to M_S^2 via m_S. Thus we can apply the Embedding Theorem with NACs [6] leading to a transformation sequence $M_S^2 \overset{t}{\Longrightarrow} M_H$ with a morphism $m' : M^1 \to M_H$ as shown in diagram (1). Then we apply the rules of our graph transformation as long as possible leading to a transformation sequence $M_H \overset{t'}{\Longrightarrow} M^2$. Since we have only nondeleting rules there is a morphism $d : M_H \to M^2$ with $t^2 \circ d = t_H$, and we define $m = d \circ m'$. The restriction of M^2 leads to the pullback (2) with pullback object M_T^2. This pullback and $t^2 \circ m \circ j^1 = inc_T \circ t_T^1$ imply that there exists a unique $m_T : M_T^1 \to M_T^2$ such that (3) commutes and $t_T^2 \circ m_T = t_T^1$, and by pullback decomposition also (3) is a pullback. Now we define $MT(m_S) = m_T$.

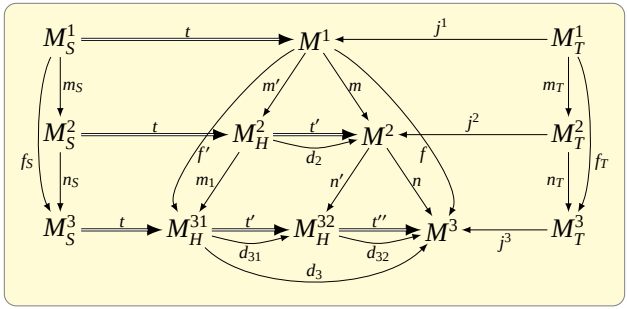


For MT to be a functor we have to show first that $MT(n_S \circ m_S) = MT(n_S) \circ MT(m_S)$ and second that $MT(id_{M_S}) = id_{MT(M_S)}$.

1. Consider the following diagram with

   $n_S \circ m_S = f_S$, $MT(m_S) = m_T$, $MT(n_S) = n_T$ and $MT(f_S) = f_T$,

   as well as the morphisms m, m', n, n' and f, f' of the construction with $d_2 \circ m' = m$, $d_{32} \circ n' = n$ and $d_3 \circ f' = f$, respectively. The functional behaviour of MT implies that we can construct the stepwise embedding $M_S^3 \overset{t}{\Longrightarrow} M_H^{31} \overset{t'}{\Longrightarrow} M_H^{32} \overset{t''}{\Longrightarrow} M^3$, since n_S is consistent w.r.t. t and m_1 is consistent w.r.t. t', and we have that $m_1 \circ m' = f'$, $n' \circ d_2 = d_{31} \circ m_1$ and $d_{32} \circ d_{31} = d_3$. Combining these results we have that $n \circ m = f$. Pullback composition and the uniqueness of pullbacks further implies that $f_T = n_T \circ m_T$, i.e. $MT(n_S \circ m_S) = MT(n_S) \circ MT(m_S)$.



2. $MT(id_{M_S}) = id_{MT(M_S)}$ follows from the functional behaviour of MT.

In the following, we present a model transformation from statecharts to Petri nets (see [3]). In Fig. 1, the integration of the type graphs for the model transformation is shown. In the left hand side, the source model type graph for statecharts is depicted. In the right hand side, the target model type graph for Petri nets is shown. Together with some additional nodes and edges they form the integrated type graph.

The rules for the model transformation are given in Figs. 2 and 3. Each state in the statechart is transformed to a corresponding place in the target Petri net model, where a token in such a place denotes that the corresponding state is active initially (rules InitState2Place and State2Place). A separate place is generated for each valid event by rule Event2Place. Each step in the statechart is transformed into a Petri net transition (rule Step2Trans). Naturally, the Petri net should simulate how to exit and enter the corresponding states in the statechart, and therefore input and output arcs of the transition have to be generated accordingly (see rules StepFrom2PreArc and StepTo2PostArc). Furthermore, firing a transition should consume the token of the trigger event (Trigger2PreArc), and

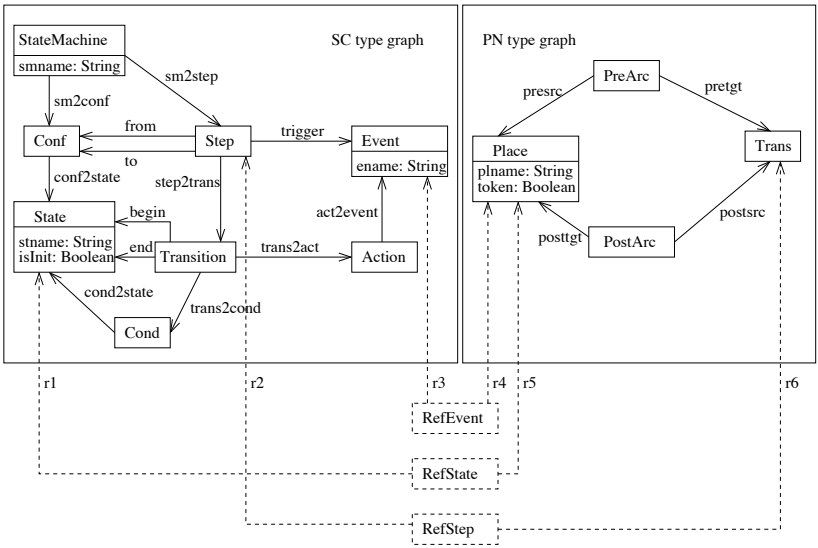


Figure 1: The integration of the type graphs

should generate tokens to (the places related to) the target event indicated as the action (Action2PostArc). All these rules are nondeleting, $TG \backslash TG_S$-generating and we have $NAC_p = \{r\}$. Moreover, the model transformation has functional behaviour (see [3]), thus we can apply the developed theory and conclude that this model transformation is a functor.

In Fig. 4, the application of the model transformation to the statecharts SC_1 and SC_2 is shown leading to the Petri nets PT_1 and PT_2, respectively. For the morphism $f_S : SC_1 \to SC_2$ we get a corresponding morphism $f_T : PT_1 \to PT_2$.

3 Conclusion

In this column we have continued the discussion of the last column concerning the claim that "model transformations should be functors". We have shown that a suitable class of model transformations based on graph transformation defines a functor between the corresponding visual languages. It remains open to analyze more general classes of model transformations and morphisms between visual models, and to show which kinds of functors can be obtained in these cases.

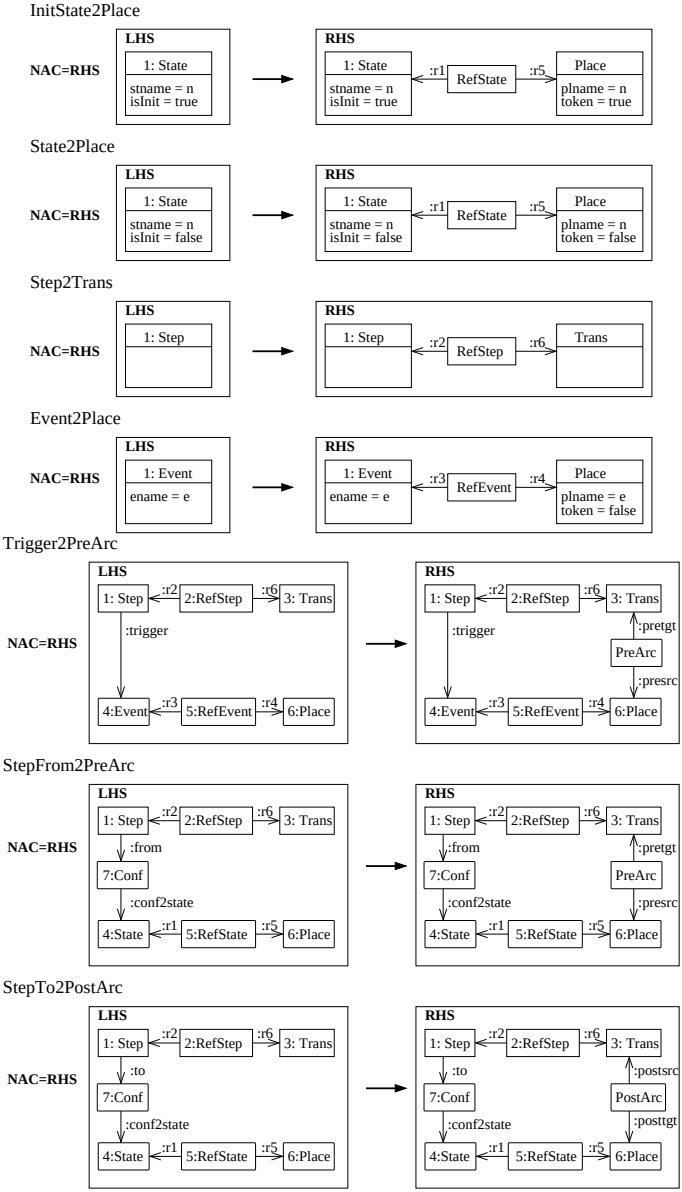


Figure 2: The rules for the model transformation (1)

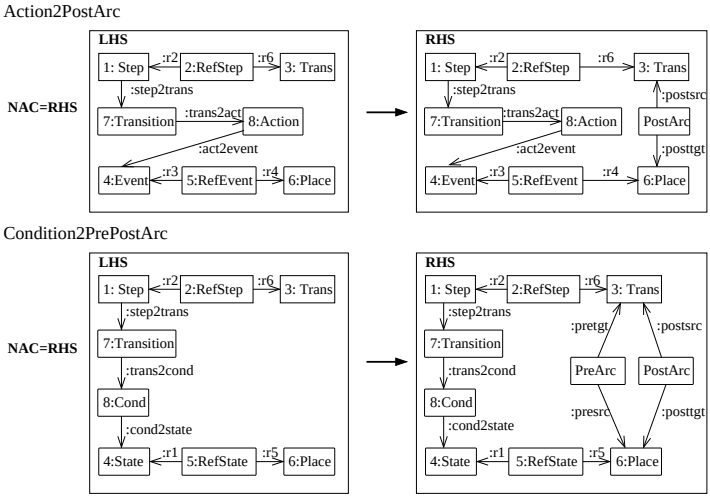


Figure 3: The rules for the model transformation (2)

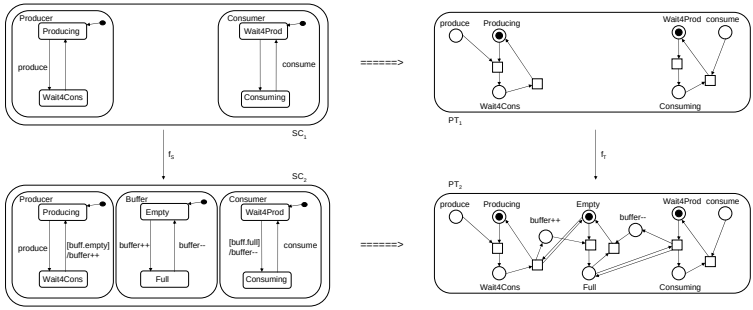


Figure 4: The model transformation and the translated morphism

References

[1] D. Batory. 2007. *From Implementation to Theory in Product Synthesis*. POPL 2007, *Keynote Speech*.

[2] D. Batory, O. Diaz, H. Ehrig, C. Ermel, U. Prange, and G. Taentzer, 2007. *Model Transformations Should Be Functors*. Bulletin of the EATCS, 92:75-81.

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

[4] H. Ehrig and B. Mahr, 1985. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of EATCS Monographs on Theoretical Computer Science, Springer.

[5] H. Ehrig and B. Mahr, 1990. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, volume 21 of EATCS Monographs on Theoretical Computer Science, Springer.

[6] L. Lambers, 2007. *Adhesive High-level Replacement Systems with Negative Application Conditions*. Technical Report 2007/14, Technical University of Berlin.