Electronic Communications of the EASST
Volume 47 (2012)

Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012)

View-based Modelling and State-Space Generation for Graph Transformation Systems

Niaz Arijo and Reiko Heckel

14 pages

Guest Editors: Andrew Fish, Leen Lambers
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

View-based Modelling and State-Space Generation for Graph Transformation Systems

Niaz Arijo[1] **and Reiko Heckel**[2]

[1] nha2@leicester.ac.uk, [2] reiko@mcs.le.ac.uk
Department of Computer Science, University of Leicester, UK

Abstract: Modelling complex systems by graph transformation, we face scalability challenges both in our ability to create and understand these models and in the ability of tools to analyse them. To address these problems we propose to model graph transformation systems in views which can be understood and analysed separately. In particular, we show that transition systems can be generated separately for different views which, when synchronised using a CSP-like operator, yield a system that is bisimilar to the original global system.

Keywords: modularity, mobility, performance modelling, graph transformation, process algebra

1 Introduction

When modelling applications in domains such as mobile, distributed, socio-technical or biological systems, we often face the requirement to represent structural dynamics as well as uncertainty of outcome or timing. Stochastic graph transformation systems address this combination. One approach to analysis of such systems [AHTG11] requires the generation of labelled transition systems (LTS) to derive a Continuous Time Markov Chains (CTMC) as input to stochastic analysis tools.

Models in the aforementioned areas are often complex, addressing different aspects of the system being modelled as well as its interaction with the environment. For example, a traffic information system (TIS) lives in an environment determined by the road network (used by vehicles), and existing wireless networks for propagating accident information. Apart from unreliable connections, applications on mobile clients have to be location-aware to deliver the desired functionality. To predict the performance of such an application, or optimise its parameters, we have to be aware of this environment and incorporate this into our models.

This complexity implies two challenges: How to structure a complex model addressing different aspects of a complex domain, and How to overcome the usual scalability problem in generating the LTS of a complex model. We address these questions by a view-based approach to modelling and state space generation. For example, in our model the system is structured into two views, one addressing location and physical mobility and the other one connectivity and communication between vehicles and the TIS. In a bottom-up approach, such views can be created separately to be composed along common interfaces to yield a global view of the system. In a top-down approach, a global system can be decomposed to allow for the local generation of LTSs and their subsequent synchronisation.

To implement the approach, we use GROOVE [Ren04] for creating graph transformation systems as views and generating their LTSs. These are then translated, using our own tool [AHTG11], into processes in PEPA, the Performance Evaluation Process Algebra [Hil05a, Hil05b] which allows us to synchronise them. PEPA is supported by an Eclipse-based environment [TDG09] giving access to a range of model evaluation and analysis tools.

In this paper we focus on the correctness of the view-based approach, describing the conditions under which the composition or decomposition of views is correct with respect to the behaviour of the global system, i.e., such that the synchronisation of the local views' LTSs yields an LTS bisimilar to that derived directly from the global system. In particular, this requires to identify conditions on rule signatures and negative application conditions to ensure that synchronisation of local steps on shared labels yields consistent global steps and that global steps can be projected into corresponding local ones. This applies to both the top down approach which starts from a global system, decomposing it, and the bottom up approach composing individual components.

After discussing related work below and recalling the basic definitions of typed attributed graph transformation in Sect. 3, Sect. 4 introduces the signatures equipped to derive labelled transition systems from graph transformation systems. Sect. 5 introduces the operations of projection and composition at specification level before Sect. 6 studies their semantic counterparts at the level of transition systems. Sect. 7 concludes the paper.

2 Related Work

Using modularity to reduce complexity is an old idea, even in graph transformation. Specifically the present approach is inspired by proposals on view based modelling [EHTE97, Hec98], but apart from being fully formalised, differs from it in two ways. First, we consider attributes and application conditions, and second, the present approach does not apply a loose semantics. The reason is that, to generate a transition system, loose semantics is not practical because it would allow far too many transitions. At the specification level this means that in this paper we use more restrictive conditions on views and their composition. The formalisation in this paper confirms the earlier, informal approach in [AHTG11] where the TIS case study is developed in more detail and analysed using tool support for projection into views and derivation of PEPA processes.

More recently, the idea of "borrowed context" has been used to achieve modularity [BEK06]. Specifically, this idea has been applied in [Ren10] to generate transition systems in GROOVE in a compositional way. This work shares some of our motivation, but achieves modularity by decomposing graphs at the instance rather than at the type level. It does not consider negative application conditions.

Outside of graph transformation, the problem of merging behavioural models has attracted attention [NC08]. Usually, the idea is to work at the level of LTS and consider merging as common refinement [UC04]. At this level, our approach appears more basic because, rather than refinement, we use composition of LTS at the same level of abstraction. However, we take into account the internal structure of states and specifications of the operations transforming them, making for a more complex model. Typical examples discussed in [NC08] include state machines and message sequence charts, i.e., high-level behavioural models without reference to complex data states.

3 Typed Attributed Graph Transformation

This section introduces the basic notions of typed attributed graph transformation with negative application conditions, illustrating them using our TIS example. The formalisation follows the algebraic approach [EEPT06]. A graph is a tuple (V,E,src,tgt) where V is a set of nodes (or vertices), E is a set of edges and $src,tgt : E \to V$ associate, respectively, a source and target node for each edge in E. Given graphs G_1 and G_2, a graph morphism is a pair (f_V,f_E) of total functions $f_V : V_1 \to V_2$ and $f_E : E_1 \to E_2$ such that source and targets of edges are preserved.

An E-graph is a graph equipped with an additional set V_A of data nodes (or values) and special sets of edges E_{EA} (edge attributes) and E_{NA} (node attributes) connecting, respectively, edges in E and nodes in V to values in V_A. An attributed graph is a tuple (EG,A) where EG is an E-graph and A is an algebra with signature $\Sigma = (S,OP)$ such that $\biguplus_{s\in S} A_s = V_A$. Intuitively, an attributed graph is an E-graph where V_A is the set of all data values available for attribution. A morphism $f : (EG,A) \to (EG',A')$ of attributed graphs is a pair of an E-graph morphism $f_{EG} : EG \to EG'$ and a compatible algebra homomorphism $f_A : A \to A'$. Fixing an attributed graph TG as *attributed type graph*, we define the category $\mathbf{AGraph_{TG}}$ of TG-typed attributed graphs [EEPT06]. Objects are pairs (G,t) of attributed graphs G with typing homomorphisms $t : G \to TG$ and morphisms $f : G \to H$ are attributed graph morphisms compatible with the typing.

We denote by $X = (X_s)_{s\in S}$ a family of countable sets of variables, indexed by sorts $s \in S$, and write $x : s \in X$ for $x \in X_s$. A TG-typed graph transformation rule (or production) over X is a span $L \xleftarrow{l} K \xrightarrow{r} R$ where l,r are monomorphisms, the algebra component of L,K,R is $T_\Sigma(X)$ (the term algebra of Σ with variables in X) such that $l_A = r_A = id_{T_\Sigma(X)}$. That means, variable names are preserved across the rule. The class of all rules over TG with variables in X is denoted $Rules(TG,X)$.

We use graphs to model the structure of the application, including the topology of locations, current locations of relevant devices, links between application components and their states. The type graph provides a structural model of the admissible states of the system, similar to the way a class diagram describes valid object structures. Fig. 1 shows the type graph of the TIS model (left) with an instance graph (right) representing a map of Roads and Junctions as well as two Cars following predefined Paths. The model allows to represent Accidents that can be reported to the TIS, to share the information with other Cars. To identify Cars, these nodes have been given *id* attributes of type integer. In general we allow graph nodes to be attributed by values of predefined data types, such as strings or natural numbers. In this paper all attribute values will be positive integers.

The operational semantics of rules is defined by the double-pushout construction. Given a TG-typed graph G and graph production $L \xleftarrow{l} K \xrightarrow{r} R$ together with a match (a TG-typed graph morphism) $m : L \to G$, a *direct derivation* $G \overset{p,m}{\Longrightarrow} H$ exists if and only if the diagram below can be constructed, where squares (1) and (2) are pushouts in $\mathbf{AGraph_{TG}}$ such that G,C,H share the same algebra A and the algebra components l^*_A, r^*_A of morphisms l^*, r^* are identities on A. This ensures that data elements are preserved across derivation sequences, which allows their use as actual parameters in a global namespace. We also write $G \overset{p/d}{\Longrightarrow} H$ to refer to the entire DPO diagram $d = (d_L,d_K,d_R)$. A derivation is a sequence $G_0 \overset{p_1,\, m_1}{\Longrightarrow} G_1 \overset{p_2,\, m_2}{\Longrightarrow} \ldots \overset{p_n,\, m_n}{\Longrightarrow} G_n$ of direct

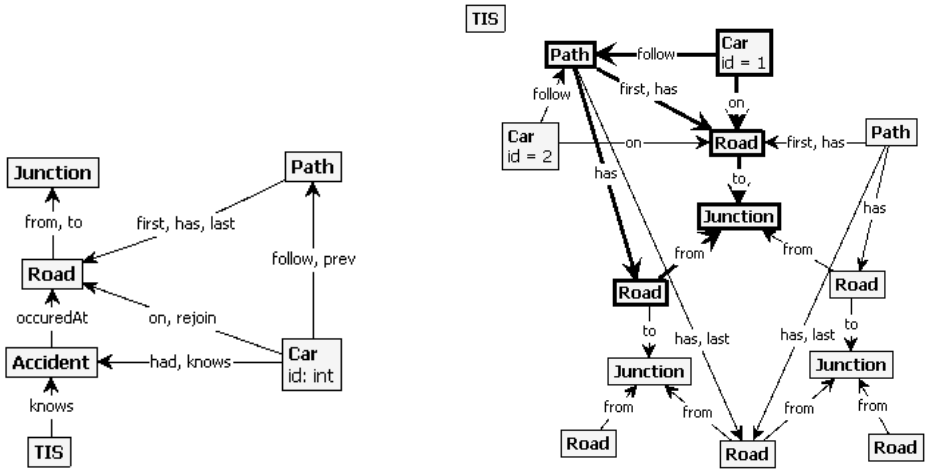


Figure 1: Type and instance graphs of a Traffic Information System (TIS).

derivations.

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m=d_L}\downarrow & (1) & \downarrow{\scriptstyle d_K} & (2) & \downarrow{\scriptstyle m^*=d_R} \\ G & \xleftarrow[l^*]{} & C & \xrightarrow[r^*]{} & H \end{array} \qquad \begin{array}{ccc} L & \xrightarrow{n} & \hat{L} \\ {\scriptstyle m}\downarrow & \swarrow{\scriptstyle q} & \\ G & & \end{array}$$

A *negative constraint* on a TG-typed graph L is a morphism $n : L \to \hat{L}$ over TG. A morphism $m : L \to G$ satisfies n (written $m \models n$) iff there is no morphism $q : \hat{L} \to G$ such that $q \circ n = m$, A negative application condition (NAC) over L is a set of negative constraints N. A morphism $m : L \to G$ *satisfies* N (written $m \models N$) if and only if m satisfies every constraint in N, i.e., $\forall n \in N : m \models n$.

In GROOVE notation [Ren04] the various components of a rule (called readers, erasers, creators or embargoes) are combined within a single rule graph, distinguishing them by different colours and styles. For example in rule *moveCar* in Fig. 2, readers in K, such as the nodes of type *Car, Path, Road, Junction*, and the edges of type *follow, has, to, from*, are thin and solid ordinary graph elements. Erasers in $L \setminus l(K)$, such as the edge of type *on* pointing to the left-most *Road* node, are shown as thin and dashed elements. Creators in $R \setminus r(K)$, such as the other edge of type *on* are represented by slightly wider solid lines. Embargoes in negative constraints, such as the nodes of type *Accident* and the edges of type *occurredAt*, *had* and *rejoin*, are represented with wider and dashed outline. Attribute values are depicted as circles pointed to by an edge from the attributed node. For example, in *moveCar* the Car node has an attribute *id* whose value is $\$1 : int$.

4 Signatures and Systems

We introduce a notion of observation on transformation steps based on rule names with parameters, whose declarations are collected in a transformation signature. Throughout the rest of the

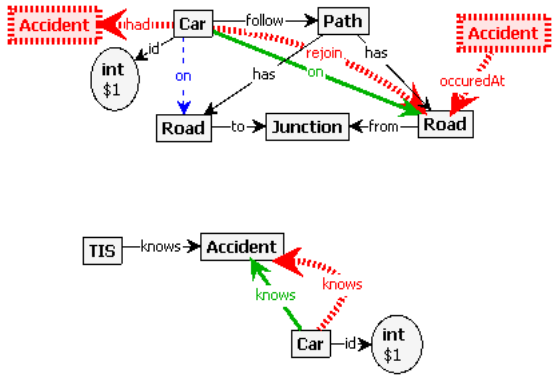


Figure 2: Top to bottom: Rules *moveCar* and *sendAccidentInfo*

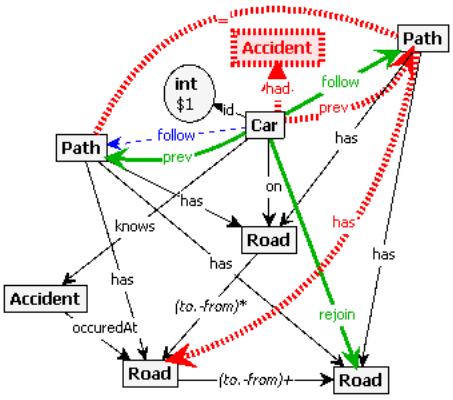


Figure 3: Rule *detour* for Car to avoid location of Accident

paper we assume a signature $\Sigma = (S, OP)$, a family $X = (X_s)_{s \in S}$ of countable sets of variables and a Σ-algebra A.

Definition 1 (transformation signature, labels) A *transformation signature* (over Σ, X, A) is a tuple $\mathscr{S} = (TG, P, \sigma)$ where

- TG is an attributed type graph,
- P is a countable set of rule names,
- $\sigma : P \longrightarrow X^*$ assigns each rule name a list of parameters $\sigma(p) = (x_1 : s_1, \ldots, x_n : s_n)$ with $x_i \in X_{s_i}$ for $1 \leq i \leq n$.

For $p \in P$ with $\sigma(p) = (x_1 : s_1, \ldots, x_n : s_n)$ we also write $p(x_1 : s_1, x_2 : s_2, \ldots, x_n : s_n) \in \mathscr{S}$.

Given a rule signature $p(x_1 : s_1, \ldots, x_n : s_n) \in \mathscr{S}$ we denote by $p(A)$ the set of all rule labels $p(a_1, \ldots, a_n)$ with $a_i \in A_{s_i}$. The label alphabet L over $\mathscr{S}$ is the union over all rule labels $\bigcup_{p \in P} p(A)$.

For example, the signatures of the rules in Fig. 2 are $moveCar(\$1 : int)$ and $sendAccidentInfo(\$1 : int)$, both referring to the *id* attribute of Car nodes. A graph transformation system over a signature adds definitions of rules with application conditions for all rule names.

Definition 2 (TAGTS) Given a transformation signature $\mathscr{S} = (TG, P, \sigma)$, a *typed attributed graph transformation system* (TAGTS) over $\mathscr{S}$ is a tuple $\mathscr{G} = (\mathscr{S}, \pi, N)$ where

- $\pi : P \longrightarrow Rules(TG, X)$ assigns each rule name a span $sp = L \xleftarrow{l} K \xrightarrow{r} R$ over TG and X.
- $N = (N_p)_{p \in P}$ is a family of application conditions such that for $\pi(p)$ as above, N_p is a NAC over L.

Observations on transformations are defined by $obs(G \overset{p,m}{\Longrightarrow} H) = p(a_1, a_2, \ldots, a_n)$ if $\sigma(p) = (x_1 : s_1, \ldots, x_n : s_n)$ and $a_i = m(x_i)$. A step $t = (G \overset{p,m}{\Longrightarrow}_{\mathscr{G}} H)$ is a transformation such that match m

satisfies N_p. A derivation is a sequence $G_0 \overset{p_1,m_1}{\Longrightarrow}_{\mathscr{G}} \ldots \overset{p_n,m_n}{\Longrightarrow}_{\mathscr{G}} G_n$, also written $G_0 \Longrightarrow^*_{\mathscr{G}} G_n$. The pair $(\mathscr{G}, G_0)$ with G_0 a TG-typed graph is called a *typed attributed graph grammar (TAGG)* over $\mathscr{S}$. Grammar $(\mathscr{G}, G_0)$ is *deterministic* if for all reachable graphs G and transformations via $p \in P$, $\mathrm{obs}(G \overset{p,m}{\Longrightarrow}_{\mathscr{G}} H) = \mathrm{obs}(G \overset{p,m'}{\Longrightarrow}_{\mathscr{G}} H')$ implies $m = m'$.

Spans are attributed over $T_\Sigma(X)$ and parameters $x_i \in X_{s_i}$ are from the same set, so they refer to the variables used in attribute expressions in rules. The algebra component of attributed graphs is preserved by transformations, allowing actual parameters to be used globally.

Grammars are deterministic if labels carry enough information to determine the match into any graph reachable from the start graph. While the notion abstracts from the mechanism by which this happens, we employ *id* attributes on nodes, referred to by rule parameters as part of transformation labels. If, as in our example, we do not have parallel edges and all relevant nodes are equipped with unique ids, matches can be determined by listing enough left-hand side nodes as parameters. Creating new nodes, we have to assign fresh *id* numbers using a counter to remember the last unused one, which is increased each time a node is created.

We will see in Sec. 6 that the notion of deterministic grammars is required for the composition of grammars to be semantically meaningful. The requirement will be imposed on the interface (intersection) only of the two grammars to be composed, rather than on the component grammars themselves.

Applying the same rule at the same match leads to isomorphic DPO diagrams and resulting graphs. The transition system generated by a TAGTS identifies isomorphic graphs, so deterministic TAGTS lead to deterministic transition systems.

Definition 3 (induced labelled transition system) Let $(\mathscr{G}, G_0)$ be a TAGG and $\mathscr{G} = (TG, P, \sigma, \pi, N)$. The *labelled transition system* $LTS(\mathscr{G}, G_0)$ is given by $(L, S, T, [G_0])$ where

- S is the set of all isomorphism classes of graphs reachable from G_0, i.e. $S = \{ [G_n] \mid G_0 \Longrightarrow^*_{\mathscr{G}} G_n \}$;
- L is the label alphabet over $\mathscr{S}$;
- $T \subseteq S \times L \times S$ is the transformation relation, where $\langle [G], l, [H] \rangle \in T$ iff there is a transformation step $t = (G \overset{p,m}{\Longrightarrow} H)$ with $obs(t) = l$;
- $[G_0]$ is the isomorphism class of the initial graph G_0.

5 Projection and Composition of Systems

We would like to distinguish two views, that of the Car and its location and mobility and that of the TIS broadcasting news about Accidents. Given a global model of the system, intuitively a view is defined by identifying in the global type graph the node and edge types that should be abstracted from, such as the TIS in the Car view and Roads and Junctions in the Service view. Start graph and rules are then reduced to the remaining types, removing all instances of types that are no longer present in the smaller type graph.

Given a subgraph $TG' \subseteq TG$, a TG-typed instance graph G can be projected to an instance $G|_{TG'}$ of TG' by removing all elements of G whose types are in TG, but not in TG'. This projection, formally the inverse image of TG' under the morphism typing G in TG, can be described

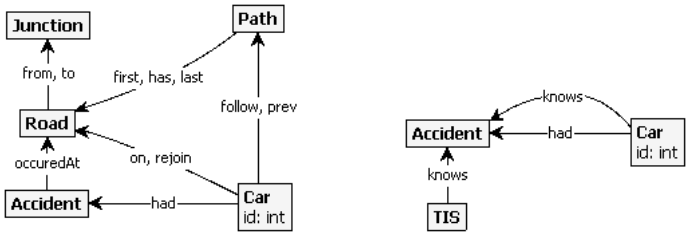


Figure 4: Type graphs of *Car* and *Service* views.

abstractly as a pullback of $TG' \subseteq TG$ and G's typing morphism. The projection extends to morphisms, making it a functor $\_|_{TG'} : \mathbf{AGraph}_{TG} \to \mathbf{AGraph}_{TG'}$ [HEET99].

The functor can be used to define the projection of graph transformation rules and application conditions to a subgraph of their current type graph. We define a more general projection based on a subsignature, which allows to reduce the set of rule names as well as the type graph.

Definition 4 (subsignature, projection) A *transformation signature* $\mathscr{S}' = (TG', P', \sigma')$ is a subsignature of $\mathscr{S} = (TG, P, \sigma)$, written $\mathscr{S}' \subseteq \mathscr{S}$, iff $TG' \subseteq TG$, $P' \subseteq P$, and $\sigma' = \sigma|_{P'}$.

Given a TAGTS $\mathscr{G} = (\mathscr{S}, \pi, N)$ we define the restriction $\mathscr{G}|_{\mathscr{S}'} = (\mathscr{S}', \pi', N')$ of $\mathscr{G}$ to $\mathscr{S}'$ for all $p \in P'$ by

- $\pi'(p) = L|_{TG'} \xleftarrow{l|_{TG'}} K|_{TG'} \xrightarrow{r|_{TG'}} R|_{TG'}$ if $\pi(p) = L \xleftarrow{l} K \xrightarrow{r} R$
- $N'_p = \{n|_{TG'} : L|_{TG'} \to \hat{L}|_{TG'} \mid n : L \to \hat{L} \in N_p$ s.t. the diagram below is a pushout.$\}$

$$\begin{array}{ccc} \hat{L}|_{TG'} & \xleftarrow{n|_{TG'}} & L|_{TG'} \\ \downarrow & & \downarrow \\ \hat{L} & \xleftarrow{n} & L \end{array}$$

The projection is *sound* if for all rules $p \in P \setminus P'$, it reduces $\pi(p)$ to a span of isomorphisms. A constraint n such that the diagram above forms a pushout is *preserved by the projection* to TG'.

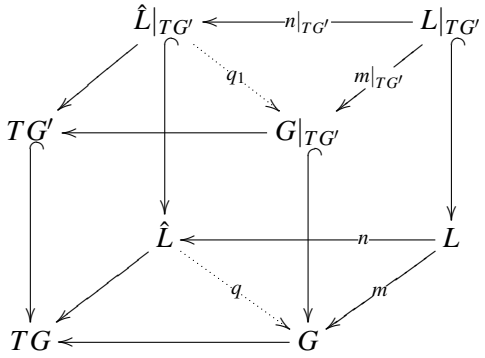
Projecting the global TIS model to the type graphs of the Car and Service views in Fig. 4, rule *moveCar* is kept unchanged in the Car view, but reduced to an idle span of isomorphisms in the Service view. Rule *sendAccidentInfo* is idle in the Car view, but kept unchanged in the Service view. Rule *detour* looses its *knows* edge in the Car view and is reduced to an idle span in the Service view. In the last case, both projections will be required for recreating the original, but this is an issue of (de)composition, not projection.

For a projection to be sound, i.e., to preserve transformations, we must only drop rules that are rendered idle, such as *moveCar, detour* from the Service view signature and *sendAccidentInfo* from the Car view signature. Constraints of retained rules are dropped if they are not preserved, i.e., if they loose any negative elements $\hat{L} \setminus n(L)$. For example, if we should decide to keep *moveCar* in the Service view, which would give a sound, if slightly redundant projection, this rule would loose both the *rejoin* and the *Accident occurredAt* constraints, while keeping the *had Accident* constraint where all types are kept in the Service view.

Lemma 1 (projection of steps) *Assume signatures and systems $\mathscr{S}' \subseteq \mathscr{S}$ and $\mathscr{G}$ as above such that the projection $\mathscr{G}' = \mathscr{G}|_{\mathscr{S}'}$ is sound. Then, for each step $t = (G \overset{p,m}{\Longrightarrow}_{\mathscr{G}} H)$, either there exists a step $t|_{\mathscr{S}'} = (G|_{TG'} \overset{p,m|_{TG'}}{\Longrightarrow}_{\mathscr{G}'} H|_{TG'})$ with $obs(t) = obs(t|_{\mathscr{S}'})$, or $G|_{TG'}, H|_{TG'}$ are isomorphic.*

Proof. If $p \in P \setminus P'$ the projection of $\pi(p)$ is a span of identities. Since **AGraph** is adhesive, the projection functor preserves pushouts where one of the given morphisms is mono, such as the pushouts in a double-pushout diagram. That means, the projection of double pushout t is a DPO over TG' where the top span is made of identities. Since pushouts preserve isos, graphs $G|_{TG'}, H|_{TG'}$ at the bottom of the DPO are isomorphic.

If $p \in P'$ we have $t|_{\mathscr{S}'}$ because (1) the projection functor preserves double pushouts and (2) $m|_{TG'} \models N'_p$ iff $m \models N_p$. To see (2), consider the cube diagram below, whose back face represents the pushout of Def. 4.1 while the sides and front are pullbacks arising from the projection functor. The back square, with its two opposing monos, is a pullback too. Using the pushout property of the back square it follows that existence of q_1 commuting the top triangles implies existence of q commuting the bottom triangles and the resulting diagonal vertical square. The reverse implication can be established with the help of the pullback property of the front square. That means, m satisfies n iff $m|_{TG'}$ satisfies $n|_{TG'}$.

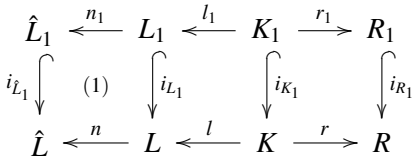


□

Composition of systems is based on composition of rules, defined by the union of two rules agreeing on a common projection. Constraints of the two given rules have to be transferred to the enlarged left-hand side of the new rule.

Definition 5 (extension and union) Assume rule span $s_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ embedded into $s = (L \xleftarrow{l} K \xrightarrow{r} R)$ by inclusions $i_{L_1}, i_{K_1}, i_{R_1}$ as in the diagram below, denoted $s_1 \subseteq s$.

If n_1 is a constraint over L_1, the extension of n_1 to L is defined by the pushout (1) where $i_{\hat{L}_1}$ is the inclusion of $\hat{L}_1$ into $\hat{L}$ and n is the extended constraint over L. If N_1 is a set of constraints over L_1, we denote by N_1^L the corresponding set of constraints extended to L.



The union of two rule spans s_1 and s_2 with $s_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ is defined if they have a well-defined intersection, that is, the componentwise intersection $s_1 \cap s_2 = L_1 \cap L_2 \xleftarrow{l_1 \cap l_2} K_1 \cap K_2 \xrightarrow{r_1 \cap r_2} R_1 \cap R_2$ is a rule span. In this case, their union is $s_1 \cup s_2 = L_1 \cup L_2 \xleftarrow{l_1 \cup l_2} K_1 \cup K_2 \xrightarrow{r_1 \cup r_2} R_1 \cup R_2$ where $X_1 \cup X_2$ is the pushout of X_1, X_2 over $X_1 \cap X_2$ with inclusions as morphisms, for $X \in \{L, K, R\}$.

Extending this notion from rules to systems, we arrive at the composition of TAGTS.

Definition 6 (composition of TAGTS) Assume a signature $\mathscr{S}$ with subsignatures $\mathscr{S}_1 \subseteq \mathscr{S}$ and $\mathscr{S}_2 \subseteq \mathscr{S}$. Let $\mathscr{S}_0 = (TG_0, P_0, \sigma_0)$ be defined by component-wise intersection $TG_0 = TG_1 \cap TG_2$, $P_0 = P_1 \cap P_2$ and $\sigma_0 = \sigma_1|_{P_0} = \sigma_2|_{P_0}$.

Given two systems $\mathscr{G}_1 = (\mathscr{S}_1, \pi_1, N_1)$ and $\mathscr{G}_2 = (\mathscr{S}_2, \pi_2, N_2)$, their composition $\mathscr{G}_1 \oplus \mathscr{G}_2$ is defined if $\mathscr{G}_1|_{\mathscr{S}_0} = \mathscr{G}_2|_{\mathscr{S}_0}$. We then call the systems *composable*, and define $\mathscr{G}_1 \oplus \mathscr{G}_2 = (\mathscr{S}_1 \cup \mathscr{S}_2, \pi, N)$ by

- $\mathscr{S}_1 \cup \mathscr{S}_2 = (TG_1 \cup TG_2, P_1 \cup P_2, \sigma_1 \cup \sigma_2)$;
- $\pi(p) =$
  $\pi_1(p)$ if $p \in P_1 \setminus P_2$
  $\pi_2(p)$ if $p \in P_2 \setminus P_1$
  $\pi_1(p) \cup \pi_2(p) = L_1 \cup L_2 \xleftarrow{l_1 \cup l_2} K_1 \cup K_2 \xrightarrow{r_1 \cup r_2} R_1 \cup R_2$ if $p \in P_1 \cap P_2$
- $N_p =$
  N_{1p} if $p \in P_1 \setminus P_2$
  N_{2p} if $p \in P_2 \setminus P_1$
  $N_{1p}^{L_1 \cup L_2} \cup N_{2p}^{L_1 \cup L_2}$ if $p \in P_1 \cap P_2$

The composition of grammars $(\mathscr{G}_1, G_0^1)$ and $(\mathscr{G}_2, G_0^2)$ is defined if $\mathscr{G}_1, \mathscr{G}_2$ are composable and $G_0^0 = G_0^1|_{TG_0} = G_0^2|_{TG_0}$. In this case, $(\mathscr{G}_1, G_0^1) \oplus (\mathscr{G}_2, G_0^2) = (\mathscr{G}_1 \oplus \mathscr{G}_2, G^0)$ with $G^0 = G_0^1 \cup G_0^2$, the pushout representing the union of G_0^1, G_0^2 over G_0^0.

Hence two systems are composable if they agree on the projection to $\mathscr{S}_0$, the intersection of their signatures. The same is true of transformations in these two views, i.e., if their projections to $\mathscr{S}_0$ coincide, they gives rise to a composed transformation using the composed rule.

Lemma 2 (composition of steps) *Assume two systems $\mathscr{G}_1$ and $\mathscr{G}_2$, their projection $\mathscr{G}_0 = \mathscr{G}_1|_{\mathscr{S}_0} = \mathscr{G}_2|_{\mathscr{S}_0}$ and composition $\mathscr{G} = \mathscr{G}_1 \oplus \mathscr{G}_2$ as above. Given steps $t_i = (G_i \overset{p,m_i}{\Longrightarrow}_{\mathscr{G}_i} H_i)$ for $i = 0,1,2$ with $t_1|_{\mathscr{S}_0} = t_2|_{\mathscr{S}_0} = t_0$, there exists a step $t_1 \oplus t_2 = (G_1 \cup G_2 \overset{p,m_1 \cup m_2}{\Longrightarrow}_{\mathscr{G}} H_1 \cup H_2)$.*

Proof. Let us start by constructing the underlying DPO diagram of $t_1 \oplus t_2$ from those of the given transformations. First, observe that $t_1 \leftarrow t_0 \rightarrow t_2$ form a span of DPO diagrams with six-tuples of graph morphisms between corresponding left, right, and interface graphs at rule and transformation level relating them. Because they are obtained as projections, these morphisms form pullback diagrams wherever a square can be found. Forming the component-wise pushout $t_1 \leftarrow t_1 \oplus t_2 \rightarrow t_2$ of $t_1 \leftarrow t_0 \rightarrow t_2$, it follows from adhesiveness that the object obtained is a double pushout and all new squares are pullbacks, too. This is a variant of the Distribution Theorem for graph transformation [EHK+97]. It remains to show that the combined DPO forms a step in $\mathscr{G}$,

i.e., that the constraints of the combined rule are satisfied. By Def. 6 these are given by extending constraints of the two component rules. Thus, by the same argument as in the proof of Lemma 1 and the fact that they are satisfied for t_1 and t_2 it follows that they are satisfied for $t_1 \oplus t_2$. □

6 Operations on Transition Systems

With the semantics of a TAGTS defined as its induced labelled transition system, we interpret their composition by a corresponding notion of composition of LTS, synchronising transitions with shared labels while interleaving those whose labels only occur in one or the other LTS. This matches the PEPA coordination operator [Hil05b] which itself is based on composition in CSP.

Definition 7 (composition of transition systems) Given $LTS_1 = (S_1, L_1, \Longrightarrow_1, s_1^0)$ and $LTS_2 = (S_2, L_2, \Longrightarrow_2, s_2^0)$, their product $LTS_1 \otimes LTS_2 = (S, L, \Longrightarrow, s^0)$ has as states S all pairs of states (s_1, s_2) with $s_i \in S_i$. The set of labels is defined by $L = L_1 \cup L_2$ and the transition relation is the smallest one satisfying

- if $l \in L_1 \setminus L_2$ and $s_1 \overset{l}{\Longrightarrow}_1 s_1'$, then $(s_1, s_2) \overset{l}{\Longrightarrow} (s_1', s_2)$;
- if $l \in L_2 \setminus L_1$ and $s_2 \overset{l}{\Longrightarrow}_2 s_2'$, then $(s_1, s_2) \overset{l}{\Longrightarrow} (s_1, s_2')$;
- if $l \in L_1 \cap L_2$, $s_1 \overset{l}{\Longrightarrow}_1 s_1'$ and $s_2 \overset{l}{\Longrightarrow}_2 s_2'$, then $(s_1, s_2) \overset{l}{\Longrightarrow} (s_1', s_2')$.

The initial state s^0 is (s_1^0, s_2^0), the pair of initial states of the two systems.

Under suitable assumptions, composition of TAGTS is reflected by the corresponding composition of their LTS. This supports the bottom-up approach of our methodology.

Proposition 1 (composition) *Assume grammars $(\mathscr{G}_1, G_1^0)$ and $(\mathscr{G}_2, G_2^0)$, with $\mathscr{G}_i = (\mathscr{S}_i, \pi_i, N_i)$ and $\mathscr{S}_0 = \mathscr{S}_1 \cap \mathscr{S}_2 = (TG_0, P_0, \sigma_0)$, such that the grammar's composition is defined,*

1. *$(\mathscr{G}_1|_{\mathscr{S}_0}, G_1^0|_{TG_0})$ is deterministic, and*
2. *for all rules $p_1 \in P_1 \setminus P_2$ (resp., $p_2 \in P_2 \setminus P_1$), the projection $\pi_1(p_1)|_{TG_0}$ (resp., $\pi_2(p_2)|_{TG_0}$) is a span of isomorphisms.*

Then, transition systems $LTS(\mathscr{G}, G^0)$ and $LTS(\mathscr{G}_1, G_1^0) \otimes LTS(\mathscr{G}_2, G_2^0)$ are bisimilar.

Proof. We write $[G_1, G_2] \sim [G]$ iff $G_i = G|_{TG_i}$ for $i = 1, 2$. States of the composed $LTS(\mathscr{G}_1, G_1^0) \otimes LTS(\mathscr{G}_2, G_2^0)$ are pairs of isomorphism classes $([G_1], [G_2])$, denoted as $[G_1, G_2]$. States of $LTS(\mathscr{G}, G^0)$ are isomorphism classes of graphs $[G]$. We will show that relation $\sim$ as defined above is a bisimulation between $LTS(\mathscr{G}_1, G_1^0) \otimes LTS(\mathscr{G}_2, G_2^0)$ and $LTS(\mathscr{G}, G^0)$. Recall that a bisimulation is a relation R between the states of two systems such that, if $P_1 \; R \; P_2$, then both $P_1 \overset{l}{\longrightarrow}_1 Q_1$ implies $P_2 \overset{l}{\longrightarrow}_2 Q_2$ and $Q_1 \; R \; Q_2$ as well as $P_2 \overset{l}{\longrightarrow}_2 Q_2$ implies $P_1 \overset{l}{\longrightarrow}_1 Q_1$ and $Q_1 \; R \; Q_2$.

Assume that $[G_1, G_2] \sim [G]$ and let $[G] \overset{l}{\longrightarrow} [H]$ be a transition in $LTS(\mathscr{G}, G^0)$. That means, there exists a transformation $t = (G \overset{p,m}{\Longrightarrow}_{\mathscr{G}} H)$ with $\mathrm{obs}(t) = l$. The projections $\mathscr{G}|_{\mathscr{S}_i}$ are sound because of Assumption 2 in the proposition above.

Hence, with Lemma 1, $G_i = G|_{TG_i}$ and $H_i = H|_{TG_i}$, there are either two projections $t_i = (G_i \overset{p,m|_{TG_i}}{\Longrightarrow}_{\mathscr{G}_i} H_i)$ with $\mathrm{obs}(t) = \mathrm{obs}(t_i) = l$ for $i = 1,2$, or one such projection, say WLOG for $i = 1$, while $G_2 \cong H_2$. In the first case, $[G_i] \xrightarrow{l}_i [H_1]$ are transitions in $LTS(\mathscr{G}_i, G_i^0)$. By the third clause in Def. 7, $[G_1,G_2] \xrightarrow{l} [H_1,H_2]$ is a transition in $LTS(\mathscr{G}_1,G_1^0) \otimes LTS(\mathscr{G}_2,G_2^0)$. In the second case, $[G_1,G_2] \xrightarrow{l} [H_1,G_2] = [H_1,H_2]$ by the first clause and $[G_2] = [H_2]$.

Vice versa, let $[G_1,G_2] \xrightarrow{l} [H_1,H_2]$ be given in $LTS(\mathscr{G}_1,G_1^0) \otimes LTS(\mathscr{G}_2,G_2^0)$. By Def. 7 either there are transitions $[G_i] \xrightarrow{l} [H_i]$ in $LTS(\mathscr{G}_i,G_i^0)$ for $i = 1,2$, based on steps $t_i = (G_i \overset{p,m_i}{\Longrightarrow}_{\mathscr{G}_i} H_i)$, or there is one such transition, say WLOG for $i = 1$, and $[G_2] = [H_2]$.

In the first case, it is easy to see that the projections from $\mathscr{S}_i$ to $\mathscr{S}_0$ are sound because, e.g., $P_1 \setminus P_1 \cap P_2 = P_1 \setminus P_2$ and so soundness from $\mathscr{S}_1$ to $\mathscr{S}_0$ follows from Assumption 2 above. The projections yield $t_i|_{\mathscr{S}_0} = (G_i|_{TG_0} \overset{p,m_i|_{TG_0}}{\Longrightarrow} H_i|_{TG_0})$ with $\mathrm{obs}(t_i|_{TG_0}) = l$. $[G_1,G_2] \sim [G]$ implies $G_1|_{TG_0} = G_2|_{TG_0} =: G_0$. Since $(\mathscr{G}_0,G_0^0) = (\mathscr{G}_1|_{\mathscr{S}_0}, G_1^0|_{TG_0})$ is deterministic and G_0 is reachable, $m_0 = m_1|_{TG_0} = m_2|_{TG_0}$, so $t_i|_{\mathscr{S}_0} = (G_0 \overset{p,m_0}{\Longrightarrow}_{\mathscr{G}_0} H_i|_{TG_0})$ for $i = 1,2$ are related by unique isomorphisms on intermediate and derived graphs.

To apply Lemma 2 we require steps in $\mathscr{G}_1,\mathscr{G}_2$ which project to the same step in $\mathscr{G}_0$. Using the unique isos between $t_1|_{\mathscr{S}_0}$ and $t_2|_{\mathscr{S}_0}$ and the fact that $t_2|_{\mathscr{S}_0} \subseteq t_2$ we can rename t_2 into $t_2' = (G_2 \overset{p,m_2}{\Longrightarrow}_{\mathscr{G}_2} H_2')$ such that $t_2'|_{\mathscr{S}_0} = t_1|_{\mathscr{S}_0}$. In particular, $H_1|_{TG_0} = H_2'|_{TG_0} = H_1 \cap H_2'$ and so $t_1 \oplus t_2' = (G_1 \cup G_2 \overset{p,m_1 \cup m_2}{\Longrightarrow}_{\mathscr{G}} H_1 \cup H_2')$ with $\mathrm{obs}(t_1 \oplus t_2') = l$ is a transformation in the composed TAGTS delivering transition $[G] \xrightarrow{l} [H_1 \cup H_2']$. Then, $[H_1,H_2] \sim [H_1 \cup H_2']$ because $H_2 \cong H_2'$ and therefore $[H_1,H_2] = [H_1,H_2'] \sim [H_1 \cup H_2']$.

In the second case, by Def. 7, $l \in L_1 \setminus L_2$ so by Def. 1 $p \in P_1 \setminus P_2$. By Assumption 2 in the proposition above the projection $\pi_1(p)|_{TG_0}$ yields a span of isomorphisms which extends to a transformation $t_0 = (G_1|_{TG_0} \overset{p,m_1|_{TG_0}}{\Longrightarrow} H_1|_{TG_0})$ with $G_1|_{TG_0} \cong H_1|_{TG_0}$. Any $G_2 \cong H_2$ can be extended to a DPO diagram over TG_2 that projects to t_0, and following a similar argument as above we can build $(G_1 \cup G_2 \overset{p,m_1}{\Longrightarrow}_{\mathscr{G}_1} H_1 \cup H_2)$, extending $(G_1 \overset{p,m_1}{\Longrightarrow}_{\mathscr{G}_1} H_1)$ by the idle context $G_2 \cong H_2$. Thus $[H_1,H_2] \sim [H_1 \cup H_2]$ and $[G] \xrightarrow{l} [H_1 \cup H_2]$. □

The condition for the projection to $\mathscr{S}_0$ to be deterministic ensures that, on the interface, labels uniquely determine transformations. Thus transitions carrying the same labels in the two views can be synchronised. The corollary below derives the conditions for decomposition of systems by projection.

Corollary 1 (decomposition) *Assume a system $\mathscr{G}$ over $\mathscr{S}$ with sub-signatures $\mathscr{S}_1$ and $\mathscr{S}_2$ such that $\mathscr{S} = \mathscr{S}_1 \cup \mathscr{S}_2$, the projections $\mathscr{G}|_{\mathscr{S}_1}$ and $\mathscr{G}|_{\mathscr{S}_2}$ are sound,*

1. *$\mathscr{G}|_{\mathscr{S}_1 \cap \mathscr{S}_2}$ is deterministic;*
2. *for all rules $p \in P_1 \setminus P_2$, $\pi(p)|_{TG_1} = \pi(p)$ and $N_p|_{TG_1} = N_p$ while $\pi(p)|_{TG_2}$ is a span of isomorphisms, and vice versa swapping TG_1,P_1 and TG_2,P_2;*
3. *for all $p \in P_1 \cap P_2$, any constraint $n \in N_p$ is preserved by projection to either TG_1 or TG_2.*

Then $LTS(\mathscr{G},G)$ and $LTS(\mathscr{G}|_{\mathscr{S}_1},G|_{TG_1}) \otimes LTS(\mathscr{G}|_{\mathscr{S}_2},G|_{TG_2})$ are bisimilar.

Proof. To apply Prop. 1 it suffices to show that $\mathscr{G}|_{\mathscr{S}_1} \oplus \mathscr{G}|_{\mathscr{S}_2} = \mathscr{G}$ and Assumptions 1, 2 are satisfied. Assumption 1 follows directly from Assumptions 1 above. Assumption 2 in Prop. 1 follows from Def. 4. Finally, $\mathscr{G}|_{\mathscr{S}_1} \oplus \mathscr{G}|_{\mathscr{S}_2} = \mathscr{G}$ because, due to Assumptions 2 and 3 above, rule spans and application conditions in $\mathscr{G}$ can be reconstructed from their projections.

Note that, with $TG = TG_1 \cap TG_2$, projection of graphs and rules is inverse to their union. This can be shown by formalising the union of type graphs as a van Kampen square, which extends to a cube with the union of instance graphs, such as the left-hand sides of corresponding rules, and their typing morphisms. □

Let us analyse the conditions of Cor. 1 by considering the three sample rules in Fig. 2 and 3. Specifically, condition 2 distinguishes rules like *sendAccidentInfo* and *moveCar*, both retained entirely in one view and mapped to spans of isos in the other, from rules like *detour*, also mapped to isos in the Service views, but loosing an edge of type *knows* in the *Car* view. While the reconstruction of the first two examples is trivial, with the rules being complete in one of the two views each, in the third case we require synchronisation to recover the projected edge. Hence in this case the rule must be retained in both views. Similarly, for a rule present in both views, each of its constraints must be preserved in at least one view. In the case of *detour*, all constraints remain in the *Car* view.

An interesting case beyond the scope of the example in this paper is the creation of nodes in the interface, briefly discussed in Sec. 4. Assume that the common projection of the two views contains a rule the creates a new node. By our assumption of determinism, matches for this rule into all reachable graphs are fixed by the label, which is shared with the corresponding rules in the two views. Since the match determines, up to iso, the DPO diagram, and the DPOs of views and interface are related by projection, transformations in the views starting from corresponding states and sharing the label agree on their projection to the interface. This includes the creation of the new node. For example, if the node is assigned an attribute in the intersection, the same value will be assigned in both views. Since graphs are taken up to isomorphism, structural properties (such as attributes, labels, or connections) are the only possible identifiers of nodes within these graphs, so we can claim that "the same node" is being generated by both views independently.

Finally, it is interesting to analyse the reduction in size of the state spaces produced by the two views. The table below shows the size of the global and local views as generated by GROOVE for two and three Cars, respectively. It turns out that we are unable to generate the LTS from the global view directly, but the synchronisation in PEPA of the LTS generated from the two local views produces a CTMC with 117098 states. In the first case, the ratio in size between local and global views is in the order of 1:10, in the second of 1:100.

	Global model	Car view	Service view
2 Cars	756 states	84 states	36 states
3 Cars	out of memory	936 states	1000 states

7 Conclusion

We have formalised notions of views for typed attributed graph transformation systems with rule signatures and developed operations for projection, composition and decomposition of systems

that are compatible with corresponding notions of composition of transition systems based on synchronisation over shared labels. Conceptually, composition and decomposition correspond to the top down and the bottom up methodologies of modular graph transformation systems.

The approach has been implemented based on GROOVE by providing an automated projection of graph transformation systems and a translation into stochastic analysis tools such as PEPA [AHTG11]. The formalisation in this paper confirms the earlier, more empirical approach.

In the future, we envisage language support for the bottom up approach to composing systems from views and investigate how the modularity achieved can be extended to the stochastic aspect. Another concern is the limitation of the present theory to the DPO approach, while GROOVE supports the more general SPO-like behaviour. This requires an encoding of rules that delete nodes with an unbounded number of incident edges, typically leading to additional states and transitions.

Bibliography

[AHTG11] N. Arijo, R. Heckel, M. Tribastone, S. Gilmore. Modular performance modelling for mobile applications. In Kounev et al. (eds.), *ICPE*. Pp. 329–334. ACM, 2011.

[BEK06] P. Baldan, H. Ehrig, B. König. Composition and Decomposition of DPO Transformations with Borrowed Context. In Corradini et al. (eds.), *ICGT*. Lecture Notes in Computer Science 4178, pp. 153–167. Springer, 2006.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, A. Corradini. Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach. In Rozenberg (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. Pp. 247–312. World Scientific, 1997.

[EHTE97] G. Engels, R. Heckel, G. Taentzer, H. Ehrig. A Combined Reference Model- and View-Based Approach to System Specification. *Int. Journal of Software and Knowledge Engeneering* 7(4):457–477, 1997.

[Hec98] R. Heckel. Compositional Verification of Reactive Systems specified by Graph Transformation. In *Proc. Fundamental Approaches to Software Engineering (FASE'98), Lisbon, Portugal*. LNCS 1382, pp. 138–153. Springer Verlag, 1998.

[HEET99] R. Heckel, G. Engels, H. Ehrig, G. Taentzer. A View-Based Approach to System Modelling based on Open Graph Transformation Systems. In Engels et al. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*. World Scientific, 1999.

[Hil05a] J. Hillston. Process Algebras for Quantitative Analysis. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. Pp. 239–248. IEEE Computer Society Press, Chicago, June 2005.

[Hil05b] J. Hillston. Tuning Systems: From Composition to Performance. *The Computer Journal* 48(4):385–400, May 2005. The Needham Lecture paper.

[NC08] S. Nejati, M. Chechik. Behavioural model fusion: an overview of challenges. In *Proceedings of the 2008 international workshop on Models in software engineering*. MiSE '08, pp. 1–6. ACM, New York, NY, USA, 2008.
doi:http://doi.acm.org/10.1145/1370731.1370733
http://doi.acm.org/10.1145/1370731.1370733

[Ren04] A. Rensink. The GROOVE Simulator: A Tool for State Space Generation. In Pfaltz et al. (eds.), *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*. Lecture Notes in Computer Science 3062, pp. 479–485. Springer Verlag, Berlin, 2004.
http://doc.utwente.nl/66357/

[Ren10] A. Rensink. Compositionality in Graph Transformation. In Abramsky et al. (eds.), *ICALP (2)*. Lecture Notes in Computer Science 6199, pp. 309–320. Springer, 2010.

[TDG09] M. Tribastone, A. Duguid, S. Gilmore. The PEPA Eclipse Plug-in. *Performance Evaluation Review* 36(4):28–33, Mar. 2009.

[UC04] S. Uchitel, M. Chechik. Merging partial behavioural models. In *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. SIGSOFT '04/FSE-12, pp. 43–52. ACM, New York, NY, USA, 2004.
doi:http://doi.acm.org/10.1145/1029894.1029904
http://doi.acm.org/10.1145/1029894.1029904