Semantical Correctness and Completeness of Model Transformations Using Graph and Rule Transformation

Hartmut Ehrig and Claudia Ermel

Department of Theoretical Computer Science and Software Technology
Technische Universität Berlin
ehrig@cs.tu-berlin.de, claudia.ermel@tu-berlin.de

Abstract. An important requirement of model transformations is the preservation of the behavior of the original model. A model transformation is *semantically correct* if for each simulation run of the source system we find a corresponding simulation run in the target system. Analogously, we have *semantical completeness*, if for each simulation run of the target system we find a corresponding simulation run in the source system.

In our framework of graph transformation, models are given by graphs, and graph transformation rules are used to define the operational behavior of visual models (called simulation rules). In order to compare the semantics of source and target models, we assume that in both cases operational behavior can be defined by simulation rules. The model transformation from source to target models is given by another set of graph transformation rules. These rules are also applied to the simulation rules of the source model. The main result in this paper states the conditions for model and rule transformations to be semantically correct and complete. The result is applied to analyze the behavior of a model transformation from a domain-specific visual language for production systems to Petri nets.

1 Introduction

In recent years, visual models represented by graphs have become very popular in model-based software development. The shift of paradigm from pure programming to visual modeling and model-driven development (MDD) led to a variety of domain-specific modeling languages (DSMLs) on the one hand, but also to a wide-spread use of general diagrammatic modeling languages such as UML and Petri nets. DSMLs provide an intuitive, yet precise way in order to express and reason about concepts at their natural level of abstraction. Starting with a domain-specific model, *model transformation* is the key technology of MDD and serves a variety of purposes, including the refinements of models, their mapping to implementations and/or semantic domains, consistency management and model evolution. For example, a complete design and analysis process involves designing the system using the design language, transforming it into the analysis language, and performing the verification and analysis on the analysis model.

H. Ehrig et al. (Eds.): ICGT 2008, LNCS 5214, pp. 194–210, 2008.

In such a scenario, it is very important that the transformation preserves the semantics of the design model.

In this paper we study semantical correctness and completeness of model transformations provided that the source and the target languages have already a formal semantics. Approaches exist where semantic equivalence between one source model and its transformed target model is shown using bisimulation. In the approach of Karsai et al. [1] the particular transformation resulted in an output model that preserves the semantics of the input model with respect to a particular property. However, analogously to syntactical correctness proofs, it is desirable to have a more general concept for showing semantical correctness and completeness of a model transformation, independent of concrete source models.

This paper discusses an approach to verify semantical correctness of model transformations on the level of model transformation rules. Basically, semantical correctness of a model transformation means that for each simulation sequence of the source system we find a corresponding simulation sequence in the target system. Vice versa, we have semantical completeness, if for each simulation sequence in the target system there is a corresponding sequence simulating the source model. In order to compare the semantics of the source and target models, we assume that in both cases operational behavior can be defined by simulation graph rules. We then apply the model transformation to the simulation rules of the source model, leading to a so-called *rule transformation*. The resulting rules are compared to the given simulation rules of the target language.

The main result in this paper states the conditions for model and rule transformations to be semantically correct and complete. The paper generalizes and extends results from *simulation-to-animation model and rule transformation* (*S2A* transformation), which realizes a consistent mapping from simulation steps in a behavioral modeling language to animation steps in a more suitable domain-specific visualization [2,3,4]. The result is applied to analyze the behavior of a model and rule transformation from a domain-specific language for production systems to Petri nets.

The structure of the paper is as follows: In Section 2, our running example, a domain-specific visual language for production systems, is introduced. Section 3 reviews the basic concepts of model and rule transformation based on graph transformation. In Section 4, the notions *semantical correctness* and *semantical completeness* of model transformations are formally defined, and conditions are elaborated for correct and complete model transformations defined by graph rules. The main result is applied to our running example, showing that the model and rule transformation from production systems to Petri nets is semantically correct and complete. The result of the rule transformation is compared with the given simulation rules of the target language of Petri nets. Section 5 discusses related work, and Section 6 concludes the paper. Please note that the long version of this paper, the technical report [5], contains the complete case study as well as all definitions and full proofs of the theorems presented in this paper.

2 Example: Simulation of Production Systems

In this section we provide a description of a DSML for production systems and its operational semantics using graph transformation rules (a slightly simplified version of the DSML presented in [6]). Note that the rules are shown in concrete syntax, thus making the expression of operational semantics intuitive and domain-specific. Fig. 1 shows in the upper part a type graph for the production system language. The language contains different kinds of machines, which can be connected through conveyors. Human operators are needed to operate the machines, which consume and produce different types of pieces from/to conveyors. Conveyors can also be connected. The lower part of Fig. 1 shows a production system model (a graph typed over the type graph above) using a visual concrete syntax. The model contains six machines (one of each type), two operators, six conveyors and four pieces. Machines are represented as boxes, except generators, which are depicted as semi-circles with the kind of piece they generate written inside. Operators are shown as circles, conveyors as lattice boxes, and each kind of piece has its own shape. Two operators are currently operating a generator of cylindrical pieces and a packaging machine respectively.

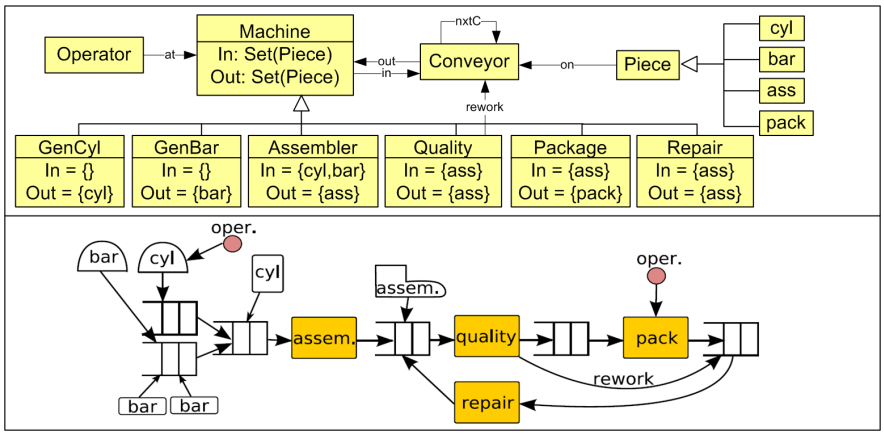


Fig. 1. Type Graph for Producer Systems and Instance Graph

Fig. 2 shows some of the graph transformation rules that describe the operational semantics for production systems. Rule assemble specifies the behaviour of an assembler machine, which converts one cylinder and a bar into an assembled piece. The rule can be applied if every specified element (except those marked as $\{new\}$) can be found in the model. When such an occurrence is found, then the elements marked as $\{del\}$ are deleted, and the elements marked as $\{new\}$ are created. Note that even if we depict rules using this compact notation, we use the DPO formalization in our graph transformation rules. In practice, this means that a rule cannot be applied if it deletes a node but not all its adjacent edges. In addition, we consider only injective matches. Rule genCylinder models

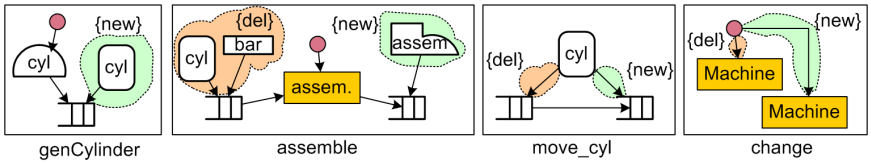


Fig. 2. Some Simulation Rules for Production Systems

the generation of a piece of kind *cylinder* which requires that the cylinder generator machine is attended by an operator and connected to a conveyor. Rule move_cyl describes the movement of cylinder pieces through conveyors. Finally, rule change models the movement of an operator from one machine (of any kind) to another one. Note that we may use abstract objects in rules (e.g., Machine is an abstract node type). In this case, the abstract objects in a rule are instantiated to objects of any concrete subclass [7]. Additional rules (not depicted) model the behaviour of the other machine types.

3 Basic Concepts of Model and Rule Transformation

In this section, we define model transformation by graph and rule transformation based on visual language specifications as typed graph transformation systems.

3.1 Visual Languages and Simulation

We use typed algebraic graph transformation systems (TGTS) in the double-pushout-approach (DPO) [8] which have proven to be an adequate formalism for visual language (VL) modeling. A VL is modeled by a type graph capturing the underlying visual alphabet, i.e. the symbols and relations which are available. Sentences or diagrams of the VL are given by graphs typed over the type graph. We distinguish abstract and concrete syntax in alphabets and models, where the concrete syntax includes the abstract symbols and relations, and additionally defines graphics for their visualization. Formally, a VL can be considered as a subclass of graphs typed over a type graph TG in the category $\mathbf{Graphs_{TG}}$.

For behavioral diagrams, an operational semantics can be given by a set of simulation rules P_S, using the abstract syntax of the modeling VL, defined by simulation type graph TG_S. A simulation rule $p = (L \leftarrow K \rightarrow R) \in P_S$ is a TG_S-typed graph transformation rule, consisting of a left-hand side L, an interface K, a right-hand side R, and two injective morphisms. In the case $L = K$, the rule is called *non-deleting*. Applying rule p to a graph G means to find a match of $L \xrightarrow{m} G$ and to replace the occurrence $m(L)$ of L in G by R leading to the target graph G'. Such a graph transformation step is denoted by $G \overset{(p,m)}{\Longrightarrow} G'$, or simply by $G \Rightarrow G'$. In the DPO approach, the deletion of $m(L)$ and the addition of R are described by two pushouts (a DPO) in the category $\mathbf{Graphs_{TG}}$ of typed graphs. A rule p may be extended by a set of *negative application conditions (NACs)* [8], describing situations in which the rule should not be applied to G. Formally,

match $L \xrightarrow{m} G$ satisfies NAC $L \xrightarrow{n} N$ if there does not exist an injective graph morphism $N \xrightarrow{x} G$ with $x \circ n = m$. A sequence $G_0 \Rightarrow G_1 \Rightarrow ... \Rightarrow G_n$ of graph transformation steps is called *transformation* and denoted as $G_0 \overset{*}{\Rightarrow} G_n$. A transformation $G_0 \overset{*}{\Rightarrow} G_n$, where rules from P are applied as long as possible, (i.e. as long as matches can be found satisfying the NACs), is denoted by $G_0 \overset{P\ !}{\Longrightarrow} G_n$. We regard a model's *simulation language* VL_S, typed over simulation alphabet TG_S, as a sublanguage of the modeling language VL, so that all diagrams $G_S \in VL_S$ represent different states of the same model during simulation. Based on VL_S, the operational semantics of a model is given by a *simulation specification.*

Definition 1 (*Simulation Specification*). *Given a visual language VL_S typed over TG_S, i.e. $VL_S \subseteq \mathbf{Graphs_{TG_S}}$, a* simulation specification *$SimSpec_{VL_S} = (VL_S, P_S)$ over VL_S is given by a typed graph transformation system (TG_S, P_S) so that VL_S is closed under simulation steps, i.e. $G_S \in VL_S$ and $G_S \Rightarrow H_S$ via $p_S \in P_S$ implies $H_S \in VL_S$. The rules $p_S \in P_S$ are called* simulation rules.

Example 1. The simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ for the production system consists of the visual language VL_S typed over TG_S, where TG_S is the type graph shown in the upper part of Fig. 1, P_S is the set of simulation rules partly shown in Fig. 2, and VL_S consists of all graphs that can occur in any production system simulation scenario, e.g. the instance graph shown in the lower part of Fig. 1 is one element of VL_S.

We divide a model and rule transformation from a source to a target simulation specification into two phases: in the first phase (called *S2I* transformation phase), non-deleting graph transformation rules are applied to the source model and to the source language simulation rules and add elements from the target language to the source model and rule graphs. The result is an *integrated simulation specification*, i.e. the resulting integrated model and simulation rules contain both source and target model elements. The second phase (called *I2T* transformation phase) restricts the integrated model and the integrated simulation rules to the type graph of the target language.

3.2 *S2I* Model and Rule Transformation

In order to transform a source simulation specification $SimSpec_{VL_S}$ to an integrated source-target simulation specification $SimSpec_{VL_I}$ where VL_I contains at least VL_S and VL_T, we define an *S2I* transformation $S2I = (S2I_M, S2I_R)$ consisting of a model transformation $S2I_M$, and a corresponding rule transformation $S2I_R$. The $S2I_M$ transformation applies model transformation rules from a rule set Q to each $G_S \in VL_S$ as long as possible (denoted by $G_S \overset{Q\ !}{\Longrightarrow} G_I$). The applications of the model transformation rules add symbols from the target language to the model state graphs. The resulting set of graphs G_I comprises the source-and-target integration language VL_I.

Definition 2 (*Model Transformation* $S2I_M$). *Given a simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ with VL_S typed over TG_S and a type graph TG_I,*

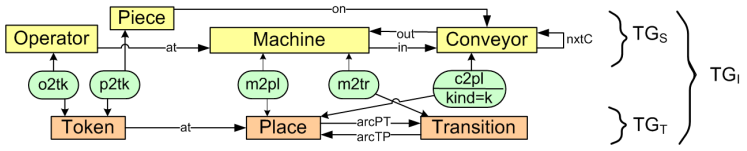


Fig. 3. Type Graph TG_I for the *ProdSystem2PetriNet* Model Transformation

called integration type graph, with $TG_S \subseteq TG_I$*, then a* model transformation $S2I_M : VL_S \to VL_I$ *is given by* $S2I_M = (VL_S, TG_I, Q)$ *where* (TG_I, Q) *is a typed graph transformation system with non-deleting rules* $q \in Q$*, and* $S2I_M$*-transformations* $G_S \overset{Q}{\Longrightarrow}{}^{!} G_I$ *with* $G_S \in VL_S$*. The* integrated language VL_I *is defined by* $VL_I = \{G_I | \exists\, G_S \in VL_S \ \wedge\ G_S \overset{Q}{\Longrightarrow}{}^{!} G_I\}$*. This means,* $G_S \overset{Q}{\Longrightarrow}{}^{!} G_I$ *implies* $G_S \in VL_S$ *and* $G_I \in VL_I$*.*

Example 2. The integrated visual language VL_I for the model transformation from production systems to Petri nets is defined by the integrated type graph TG_I in Fig. 3. The subtypes of Machine and Piece are not depicted since they are not needed in our model transformation rules. Machines and conveyors are mapped to places; pieces and operators are elements moving from one place-like element to another and hence mapped to tokens. Connections between conveyors or between machines and conveyors which indicate the way token-like elements are transported, are mapped to transitions.

The model transformation rules Q are shown in Fig. 4.

Rules *mach2place* and *conv2place* generate places for machines and conveyors. A conveyor is transformed to four different places, realizing a flattening from our model with distinct piece types to a P/T net with indistinguishable tokens.

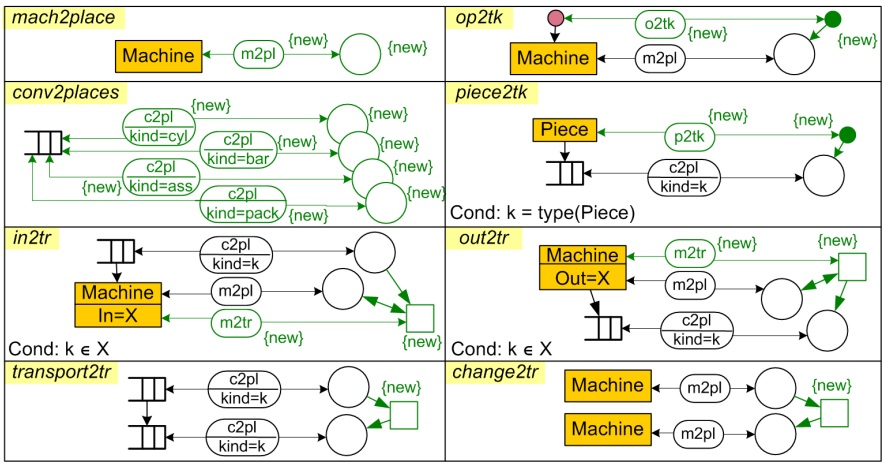


Fig. 4. *ProdSystem2PetriNet* Model Transformation Rules

Distinguishing the pieces is realized in the P/T net by placing them in distinct places. Rules *op2tk* and *piece2tk* generate tokens for operators and pieces on the places associated to their respective machines or conveyors. Transitions are generated for each connection between two conveyors (rule *transport2tr*) or between a machine and a conveyor (rules *in2tr* and *out2tr*). Two more rules (not depicted) exist which are applicable only if a machine is already connected to a transition and which just add an arc connecting the existing transition to an additional conveyor place. A machine's transition is always connected by a double arc to the machine's place to ensure that a machine is working only if an operator is present.

The result of an $S2I_M$-transformation is illustrated in Fig. 5, where a part from the model in Fig. 1 has been transformed, applying the model transformation rules in Fig. 4 as long as possible, but at most once at the same match. Our aim in this paper is not only to transform model states but to obtain a complete integrated simulation specification, including simulation rules, from the source simulation specification. In [5] we review a construction from [3,2], allowing us to apply the *S2I* transformation rules from Q also to the simulation rules, resulting in a set of integrated simulation rules. Basically, the model transformation rules are applied to each graph of a simulation rule $p_S = (L_S \leftarrow K_S \rightarrow R_S)$ as long as possible, resulting in an integrated simulation rule $p_I = (L_I \leftarrow K_I \rightarrow R_I)$. In [5] we define rule transformation for the case without NACs. An extension to NACs is given in [3,2]. Based on this definition of rule transformation, we now define an $S2I_R$ transformation of rules, leading to an *S2I* transformation $S2I = (S2I_M, S2I_R)$ from the source simulation specification $SimSpec_{VL_S}$ to the integrated simulation specification $SimSpec_{VL_I}$.

Definition 3 (*Rule Transformation* $S2I_R$). *Given a simulation specification* $SimSpec_{VL_S} = (VL_S, P_S)$ *and an* $S2I_M$*-transformation* $S2I_M = (VL_S, TG_I, Q)$*, then a* rule transformation $S2I_R : P_S \rightarrow P_I$ *is given by* $S2I_R = (P_S, TG_I, Q)$ *and* $S2I_R$ transformation sequence $p_S \overset{Q\ !}{\Longrightarrow} p_I$ *with* $p_S \in P_S$*, where rule transformation steps* $p_1 \overset{q}{\Longrightarrow} p_2$ *with* $q \in Q$ *(see [5]) are applied as long as possible. The* integrated simulation rules P_I *are defined by* $P_I = \{p_I \mid \exists\ p_S \in P_S \wedge p_S \overset{Q\ !}{\Longrightarrow} p_I\}$*. This means* $p_S \overset{Q\ !}{\Longrightarrow} p_I$ *implies* $p_S \in P_S$ *and* $p_I \in P_I$*.*

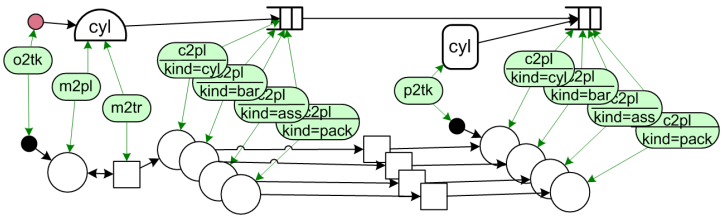


Fig. 5. *ProdSystem2PetriNet:* $S2I_M$ Model Transformation Result

Definition 4 (*S2I* Transformation, Integrated Simulation Specification). *Given* $SimSpec_{VL_S} = (VL_S, P_S)$*, an* $S2I_M$ *transformation* $S2I_M : VL_S \to VL_I$ *and an* $S2I_R$ *transformation* $S2I_R : P_S \to P_I$*, then*

1. $S2I : SimSpec_{VL_S} \to SimuSpec_{VL_I}$*, defined by* $S2I = (S2I_M, S2I_R)$ *is called* $S2I$ transformation.
2. $SimSpec_{VL_I} = (VL_I, P_I)$ *is called* integrated simulation specification*, and each transformation step* $G_I \overset{p_I}{\Longrightarrow} H_I$ *with* $G_I, H_I \in VL_I$ *and* $p_I \in P_I$ *is called* integrated simulation step.

Example 3. Fig.6 shows three integrated simulation rules, the result of $S2I_R$ transformation, i.e. of applying the model transformation rules from Fig. 4 to the source simulation rules *genCylinder*, *move_cyl* and *change* from Fig. 2.

3.3 *I2T* Transformation

In the *I2T* transformation phase, we start with the integrated simulation specification $SimuSpec_{VL_I}$ and generate the target simulation specification $SimuSpec_{VL_T}$ by restricting the integrated model graph and the integrated simulation rules to the type graph of the target language.

Definition 5 (*I2T* Transformation and Target Simulation Specification). *Given an S2I transformation* $S2I : SimSpec_{VL_S} \to SimSpec_{VL_I}$*, then*

1. $I2T: SimSpec_{VL_I} \to SimSpec_{VL_T}$*, called I2T* transformation*, is defined by* $I2T = (I2T_M : VL_I \to VL_T, I2T_R : P_I \to P_T)$ *with*
   - $I2T_M(G_I) = G_I|_{TG_T}$ *(called* $I2T_M$ *transformation), and*
   - $I2T_R(p_I) = p_I|_{TG_T}$ *(called* $I2T_R$ *transformation).*
2. $SimSpec_{VL_T} = (VL_T, P_T)$ *with* $VL_T = \{G_I|_{TG_T} \mid G_I \in VL_I\}$ *and* $P_T = \{p_I|_{TG_T} \mid p_I \in P_I\}$ *is called* target simulation specification*, and each transformation step* $G_T \overset{p_T}{\Longrightarrow} H_T$ *with* $G_T, H_T \in VL_T$ *and* $p_T \in P_T$ *is called* target simulation step.

Example 4. Fig.7 shows the target simulation rules, the result of $I2T_R$ transformation, i.e. of restricting the integrated simulation rules from Fig. 6 to the type graph of TG_T of the target language from Fig. 3 (i.e. the Petri net type graph).

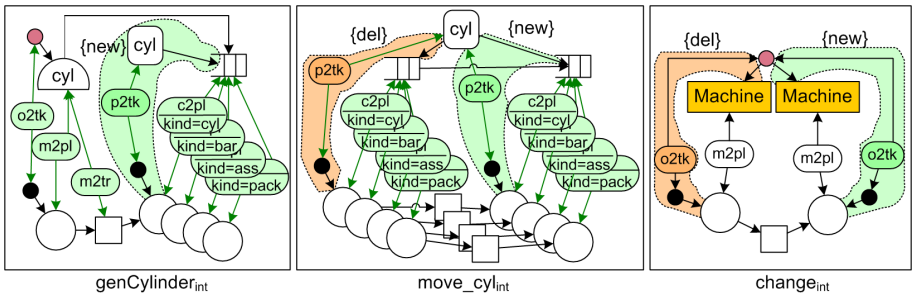


Fig. 6. Some Integrated Simulation Rules resulting from $S2I_R$ Transformation

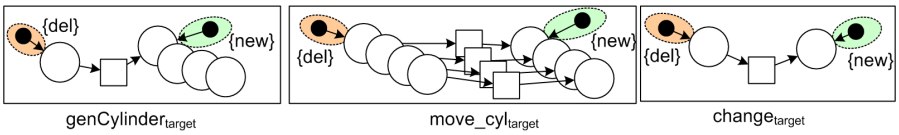


Fig. 7. Some Target Simulation Rules resulting from $I2T_R$ Transformation

We now can define the complete *S2T* model and rule transformation by combining the two transformation phases *S2I* and *I2T*.

Definition 6 (*S2T* Transformation). *Given an S2I transformation* $S2I : SimSpec_{VL_S} \to SimSpec_{VL_I}$, *and an I2T transformation* $I2T: SimSpec_{VL_I} \to SimSpec_{VL_T}$, *then* $S2T: SimSpec_{VL_S} \to SimSpec_{VL_T}$, *called S2T* transformation, *is defined by* $S2T = (S2T_M : VL_S \to VL_T, S2T_R : P_S \to P_T)$ *with*

- $S2T_M = I2T_M \circ S2I_M$ *(called* $S2T_M$ *transformation), and*
- $S2T_R = I2T_R \circ S2I_R$ *(called* $S2T_R$ *transformation).*

4 Semantical Correctness and Completeness of Model and Rule Transformations

4.1 Semantical Correctness of *S2I* Transformations

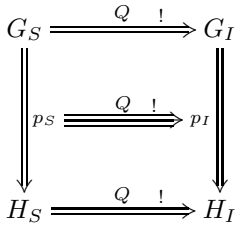
In our case, semantical correctness of an *S2I* transformation means that for each simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ there is a corresponding simulation step $G_I \overset{p_I}{\Longrightarrow} H_I$ where G_I (resp. H_I) are obtained by model transformation from G_S (resp. H_S), and p_I by rule transformation from p_S. Note that instead of a single step $G_I \overset{p_I}{\Longrightarrow} H_I$ we can also handle more general sequences $G_I \overset{*}{\Longrightarrow} H_I$ using concurrent rules and transformations. In [3], it is shown that the following properties have to be fulfilled by an *S2I*-transformation in order to be semantically correct:

Definition 7 (Termination of $S2I_M$ and Rule Compatibility of *S2I*)
An $S2I_M$ *transformation* $S2I_M : VL_S \to VL_I$ *is* terminating *if each transformation* $G_S \overset{Q}{\Longrightarrow}{}^{*} G_n$ *can be extended to* $G_S \overset{Q}{\Longrightarrow}{}^{*} G_n \overset{Q}{\Longrightarrow}{}^{*} G_m$ *such that no* $q \in Q$ *is applicable to* G_m *anymore. An S2I-transformation* $S2I = (S2I_M : VL_S \to VL_I, S2I_R : P_S \to P_A)$ *with* $S2I_M = (VL_S, TG_I, Q)$ *is called* rule compatible, *if for all* $p_I \in P_I$ *and* $q \in Q$ *we have that* p_I *and* q *are parallel and sequential independent. More precisely, for each* $G \overset{p_I}{\Longrightarrow} H$ *with* $G_S \overset{Q}{\Longrightarrow}{}^{*} G$ *and* $H_S \overset{Q}{\Longrightarrow}{}^{*} H$ *for some* G_S, $H_S \in VL_S$ *and each* $G \overset{q}{\Longrightarrow} G'$ *(resp.* $H \overset{q}{\Longrightarrow} H'$*) we have parallel (resp. sequential) independence of* $G \overset{p_I}{\Longrightarrow} H$ *and* $G \overset{q}{\Longrightarrow} G'$ *(resp.* $H \overset{q}{\Longrightarrow} H'$*).*

Theorem 1 (Semantical Correctness of *S2I*)
Given an S2I-transformation $S2I : SimSpec_{VL_S} \to SimSpec_{VL_I}$ *with* $S2I = (S2I_M : VL_S \to VL_I, S2I_R : P_S \to P_I)$ *which is rule compatible, and* $S2I_M$

is terminating. Then, S2I is semantically correct in the sense that we have for each simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ *with* $G_S \in VL_S$ *and each* $S2I_R$*-transformation sequence* $p_S \overset{Q\ !}{\Longrightarrow} p_I$ *(see Def. 3):*

1. $S2I_M$*-transformation sequences* $G_S \overset{Q\ !}{\Longrightarrow} G_I$ *and* $H_S \overset{Q\ !}{\Longrightarrow} H_I$*, and*
2. *an integrated simulation step* $G_I \overset{p_I}{\Longrightarrow} H_I$



Proof. (See [5], similar to the proof of Semantical Correctness of *S2A* in [3]).

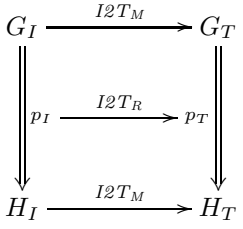
Example 5. Our *ProdSystem2PetriNet* model transformation is terminating, provided that all model transformation rules are applied at most once at each possible match. (For automatic model transformations, this can be ensured by using adequate NACs). Moreover, $S2I_R$ is rule compatible since all $p_I \in P_I$ are parallel and sequentially independent from the model transformation rules $q \in Q$. This is shown by considering all overlapping matches from a rule pair (q, p_I) into an integrated model $G_I : L_q \xrightarrow{h} G_I \xleftarrow{m} L_I$. Each overlap either is preserved by both rules, or $h(L_q)$ is completely included in $m(L_I)$. The first case is uncritical. In the second case, rule q is not applicable since it has been applied before at the same match. Hence this overlap cannot lead to a parallel dependency.

4.2 Semantical Correctness of *I2T* Transformations

We now consider the semantical correctness of the *I2T* transformation phase, which was defined in Def. 5 as the restriction of the integrated model graph and the integrated simulation rules to the type graph TG_T of the target VL.

Theorem 2 (Semantical Correctness of *I2T* Transformations). *Given an S2I transformation* $S2I = (S2I_M : VL_S \to VL_I, S2I_R : P_S \to P_I) : SimSpec_{VL_S} \to SimSpec_{VL_I}$*, and an I2T transformation* $I2T{:}SimSpec_{VL_I} \to SimSpec_{VL_T}$ *defined by* $I2T = (I2T_M, I2T_R)$ *according to Def. 5. Then, I2T is* semantically correct *in the sense that we have for each integrated simulation step* $G_I \overset{p_I}{\Longrightarrow} H_I$ *with* $G_I \in VL_I$ *and each* $I2T_R$*-transformation* $I2T_R(p_I) = p_I|_{TG_T} = p_T$*:*

1. $I2T_M(G_I) = G_T$ *and* $I2T_M(H_I) = H_T$*, and*
2. *a target simulation step* $G_T \overset{p_T}{\Longrightarrow} H_T$



Proof. See [5].

4.3 Semantical Completeness of *S2I* Transformations

In this section we consider the relation between an integrated simulation specification $SimSpec_{VL_I}$ and the corresponding source simulation specification $SimSpec_{VL_S}$. Similar to the construction of the target simulation specification

$SimSpec_{VL_T}$ by restriction of $SimSpec_{VL_I}$ to TG_T, the source simulation specification $SimSpec_{VL_S}$ can be re-constructed by restricting the integrated model graph and simulation rules to the type graph TG_S of the source language.

Definition 8 (*I2S* Backward Transformation). *Given an S2I transformation* $S2I : SimSpec_{VL_S} \to SimSpec_{VL_I}$, *then I2S:* $SimSpec_{VL_I} \to SimSpec_{VL_S}$, *called I2S* backward transformation, *is defined by* $I2S = (I2S_M : VL_I \to VL_S, I2S_R : P_I \to P_S)$ *with*

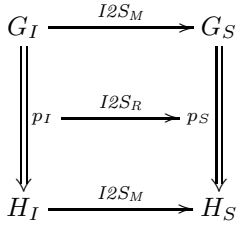
- $I2S_M(G_I) = G_I|_{TG_S}$ *(called* $I2S_M$ *backward transformation), and*
- $I2S_R(p_I) = p_I|_{TG_S}$ *(called* $I2S_R$ *backward transformation).*

The *S2I* transformation is called *faithful* if $S2I_M(G_S) = G_I$ implies $I2S_M(G_I) = G_S$ and $S2I_R(p_S) = p_I$ implies $I2S_M(p_I) = p_S$.

Remark 1. We call a rule $L \xrightarrow{q} R$ *faithful* if the restriction $q|_{TG_S}$ is the identity. It is straightforward (see [5]) to show that the *S2I* transformation is faithful if all rules $q \in Q$ are faithful.

Theorem 3 (Semantical Completeness of *S2I* Transformations). *Given a faithful S2I transformation* $S2I = (S2I_M, S2I_R) : SimSpec_{VL_S} \to SimSpec_{VL_I}$ *and its backward transformation* $I2S = (I2S_M, I2S_R) : VL_I \to VL_S$, *with* $I2S_M : VL_I \to VL_S$ *and* $I2S_R : P_I \to P_S$. *Then, S2I is* semantically complete *in the sense that we have for each integrated simulation step* $G_I \overset{p_I}{\Longrightarrow} H_I$ *with* $G_I, H_I \in VL_I$ *and* $p_I \in P_I$*:*

1. $I2S_M(G_I) = G_S$ *and* $I2S_M(H_I) = H_S$ *with* $S2I_M(G_S) = G_I$, $S2I_M(H_S) = H_I$, *and*
2. *a source simulation step* $G_S \overset{p_S}{\Longrightarrow} H_S$ *with* $I2S_R(p_I) = p_I|_{TG_S} = p_S$ *and* $S2I_R(p_S) = p_I$.



Proof. See [5].

Example 6. Our *ProdSystem2PetriNet* model transformation is faithful since all model transformation rules (see Fig. 4) add only language elements typed over $TG_I \setminus TG_S$. Hence, the rules are faithful, and the *ProdSystem2PetriNet S2I* transformation is semantically complete according to Thm. 3.

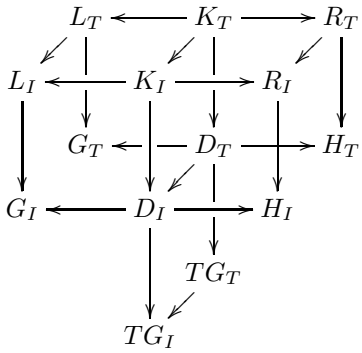
4.4 Semantical Completeness of *I2T* Transformations

Semantical completeness of *I2T* transformations means that for each simulation step in the target simulation specification we get a corresponding simulation step in the integrated simulation specification. We require the following property to be fulfilled for an *I2T* transformation in order to be semantically complete. (This property is discussed for our case study in Example 7.)

Definition 9 (*I2T* Completeness Condition)
Given a target simulation rule $p_T \in P_T$, *then due to the construction of* $SimSpec_{VL_T}$ *by restriction, there exists an integrated simulation rule* $p_I \in P_I$ *such that* $p_T =$

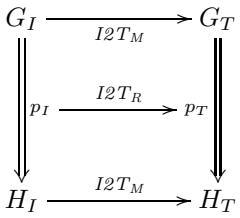
$p_I|_{TG_T}$. *Then, for each target transformation* $G_T \overset{p_T}{\Longrightarrow} H_T$ *with* $G_T \in VL_T$ *and context graph* D_T *and morphism* $K_T \to D_T$ *we require that there exists a context graph* D_I *typed over* TG_I *and morphism* $K_I \to D_I$ *such that*

1. $K_T \to D_T$ *is the restriction of* $K_I \to D_I$ *to* TG_T, *i.e. that we have two pullbacks in the diagonal squares in the diagram to the right.*
2. *For the pushout objects* G_I *and* H_I *in the front squares we have* $G_I, H_I \in VL_I$.



Theorem 4 (Semantical Completeness of *I2T*). *Each I2T transformation* $I2T = (I2T_M, I2T_R)$ *which satisfies the I2T completeness condition (see Def. 9) is semantically complete in the sense that for each target transformation* $G_T \overset{p_T}{\Longrightarrow} H_T$ *with* $G_T \in VL_T$ *via simulation rule* $p_T \in P_T$ *with* $p_T = p_I|_{TG_T}$ *for some* $p_I \in P_I$ *there is an integrated transformation* $G_I \overset{p_I}{\Longrightarrow} H_I$ *such that*

- $G_I, H_I \in VL_I$
- $G_T = G_I|_{TG_T}$ *and* $H_T = H_I|_{TG_T} \in VL_T$



Proof. See [5].

Example 7. We show that our *ProdSystem2PetriNet I2T* transformation does not fulfill the completeness condition and discuss an adaption of the model transformation rules in order to achieve satisfaction of the completeness condition.

Based on the set P_T of target rules resulting from the *ProdSystem2PetriNet I2T* transformation, we may apply more than one $p_T \in P_T$ to the same G_T. Consider for example the three target rules in Fig. 7. All of them are applicable to a target graph G_T if there exists a match from the "biggest" rule $move\_cyl_{target}$ to G_T. Thus, when applying any p_T from this set of applicable rules to G_T, we always get the same transformation span $G_T \leftarrow D_T \rightarrow H_T$, but the applied rule p_T might be the restriction of an integrated rule $p_I \in P_I$ such that the first part of the completeness condition is fulfilled, but not the second one: i.e., there exists a context graph D_I and morphism $K_I \to D_I$ such that the pushout objects G_I and H_I are *not* in VL_I. This might happen since our model transformation "forgets" information, i.e. given a target rule (typed over the Petri net language), we do not know from which integrated rule this target rule was constructed.

In order to avoid this situation, we propose a slight extension of the target type graph TG_T (Fig. 3) and the model transformation rules (Fig. 4). We introduce a suitable annotation of Petri net elements (transitions or places) by attributes keeping the information about the original role of the element. For example, we annotate each place originating from a machine by the machine type (e.g. *Assembler* or *GenCyl*), and each place originating from a conveyor by the piece type a token on this place would represent (e.g. *cyl* or *bar*). The annotation

should establish a 1:1 correspondence between the integrated rules in P_I and the target rules in P_T, and between integrated models $G_I \in VL_I$ and their target models $G_T \in VL_T$. Hence, a target rule $p_T \in P_T$ which is a restriction of an integrated rule $p_I \in P_I$ now is applicable to a target model $G_T \in VL_T$ only if there exists $G_I \in VL_I$ to which p_I is applicable. Thus, the context graph D_I and the morphism $K_I \to D_I$ are unique and lead to pushouts in the front squares such that G_I and H_I are in VL_I, i.e. also the second part of the completeness condition is now satisfied. Note that the annotation does not affect the semantical correctness and completeness of *S2I* (shown in Examples 5 and 6) since *S2I* is still terminating, rule compatible and faithful.

4.5 Semantical Correctness and Completeness of *S2T* Transformations

Putting all steps together, we find that a source-to-target model transformation *S2T*: $SimSpec_{VL_S} \to SimSpec_{VL_T}$ with $S2T = I2T \circ S2I$ is semantically correct and complete if *I2T* and *S2I* are semantically correct and complete. In this case, we get for each source simulation step in $SimSpec_{VL_S}$ a corresponding target simulation step in $SimSpec_{VL_T}$, and vice versa.

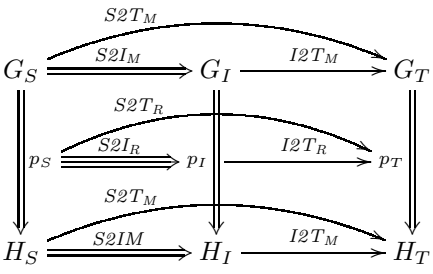
Theorem 5 (Semantical Correctness and Completeness of *S2T*). *Each S2T transformation* $S2T = (S2T_M, S2T_R) : SimSpec_{VL_S} \to SimSpec_{VL_T}$ *with* $S2T = I2T \circ S2I$, *where* $S2I : SimSpec_{VL_S} \to SimSpec_{VL_I}$ *with S2I rule compatible,* $S2I_M$ *terminating (Def. 7) and S2I faithful, and* $I2T : SimSpec_{VL_I} \to SimSpec_{VL_T}$, *with I2T satisfying the completeness condition (Def. 9), is semantically correct and complete in the following sense:*

1. Semantical Correctness: *For each source simulation step* $G_S \overset{p_S}{\Longrightarrow} H_S$ *with* $G_S \in VL_S$ *and* $S2T_R$*-transformation sequence* $p_S \overset{Q\ !}{\Longrightarrow} p_I \overset{|_{TG_T}}{\longrightarrow} p_T$ *we have*

1. $S2T_M$*-trafo* $S2T_M(G_S) = G_T : G_S \overset{Q!}{\Longrightarrow} G_I \overset{|_{TG_T}}{\longrightarrow} G_T$,
   $S2T_M$*-trafo* $S2T_M(H_S) = H_T : H_S \overset{Q!}{\Longrightarrow} H_I \overset{|_{TG_T}}{\longrightarrow} H_T$, *and*
2. *a target simulation step* $G_T \overset{p_T}{\Longrightarrow} H_T$ *via target simulation rule* $p_T \in P_T$

2. Semantical Completeness: *For each target transformation step* $G_T \overset{p_T}{\Longrightarrow} H_T$ *with* $G_T \in VL_T$ *and* $p_T \in P_T$ *there is a source simulation step* $G_S \overset{p_S}{\Longrightarrow} H_S$ *with*

- $p_T = S2T_R(p_S)$,
- $G_T = S2T_M(G_S)$ *and* $H_T = S2T_M(H_S) \in VL_T$.



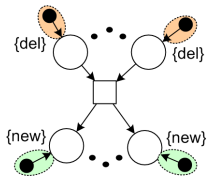
This means especially that the transformation step $G_T \overset{p_T}{\Longrightarrow} H_T$ *becomes a simulation step in* $SimSpec_{VL_T}$, *generated from the simulation step* $G_S \overset{p_S}{\Longrightarrow} H_S$.

Proof. By semantical correctness of *S2I* and *I2T* (Theorems 1 and 2), we get directly the semantical correctness of $S2T = I2T \circ S2I$. By semantical completeness of *S2I* and *I2T* (Theorems 3 and 4), we get directly the semantical completeness of $S2T = I2T \circ S2I$.

4.6 Relationship of $SimSpec_{VL_T}$ and Target Language Semantics

In the case that the target language has already an operational semantics given by simulation rules $P_{\bar{T}}$ (like in our running example, where the target language is the language of Petri nets), we may require for our model transformation *S2T* to be *behavior-preserving* in the sense that for each model in VL_T the simulations via rules in P_T correspond to simulations via rules in $P_{\bar{T}}$ and vice versa.

Example 8. As classical semantics of a P/T net (with fixed arc weight 1) we generate for each transition with i input places and o output places in a given Petri net model a corresponding firing rule [9]. Such firing rules belong to the rule schema depicted to the right. For a transition with i input places and o output places there is the graph rule $p_{\bar{T}} \in P_{\bar{T}}$ where the transition with its environment is preserved by the rule, all (and only the) input places are marked each by one token in the left-hand side, and all (and only the) output places are marked each by one token in the right-hand side.



Furthermore, the rules must not be applied to transitions with larger environment which can be ensured by suitable NACs (called environment-preserving). Considering the target simulation rules P_T which resulted from our extended *ProdSystem2Petri S2T* transformation (i.e. the rules in Fig. 7, extended by annotations as described in Example 7), we notice two differences to $P_{\bar{T}}$:

1. The target rules in P_T have no environment-preserving NACs,
2. The Petri net elements in the target rules in P_T are annotated,
3. The target rules in P_T in general contain *context* in addition to the environment of a single transition.

In case 1, we add environment-preserving NACs to each target rule without changing their applicability, since the annotations ensure that each target rule can be applied to a transition with fixed environment, anyway.

In case 2, we omit the annotations in the target rules and argue that the rules without annotations (but with environment-preserving NACs) lead to the same transformations as before. We find that all target rules without annotations which are applicable to G_T at matches overlapping in the enabled transition and its environment have the same transformation span $G_T \leftarrow D_T \rightarrow H_T$ (they are semantically equivalent) (like e.g. the target rules in Fig. 7 which are semantically equivalent for a match from the "biggest" rule $move\_cyl_{target}$ to G_T). It can be checked easily that we have a similar situation for all other target rules. The NACs prevent that target rules without annotations are applied to transitions with a larger environment. All semantically equivalent target rules without

annotations which are applicable at matches containing the same enabled transition, correspond to exactly one application of an annotated target rule at this match. Thus, we can omit the annotations in the target rules without causing changes of the possible target transformation steps.

In case 3, the behavior is preserved only if the additional context in each rule $p_T \in P_T$ can always be found for each match into any model in $SimSpec_{VL_T}$, and if this context is not changed by the rule. We have additional context for instance in the rules $genCylinder_{target}$ and $move\_cyl_{target}$ (see Fig. 7). Here, the context was generated due to the flattening of conveyors to sets of four places. Since this flattening was also performed for each conveyor in the source model G_S, we know that each match at which the rule $genCylinder_{target}$ without the three additional context places is applicable, corresponds to a match of the rule with context. This is true in our example for all firing rules containing context in addition to the active transition's environment. Hence, we can conclude that the $ProdSystem2Petri_{annotated}$ model transformation is not only semantically correct and complete, but also behavior-preserving w.r.t. the Petri net semantics.

5 Related Work

Results concerning the correctness of model transformations have been published so far mainly on formally showing the *syntactical correctness* [10].

To ensure the semantical correctness of model transformations, Varró et al. [11] use graph transformation to specify the dynamic behavior of systems and generate a transition system for each model. A model checker verifies certain dynamic consistency properties by checking the source and target models. In [1], a method is presented to verify semantical equivalence for particular model transformations. It is shown by finding bisimulations that a target model preserves the semantics of the source model w.r.t. a particular property. This technique does not prove the correctness of the model transformation rules in general.

In [2,3,4], we consider *S2A* transformation, realizing a consistent mapping from *simulation* steps in a behavioral modeling language to *animation* steps in a more suitable domain-specific visualization. The animation specification A in [2,3,4] corresponds to an integrated simulation specification in this paper. However, there is no *I2T* transformation considered in [2,3,4]. This paper generalizes and extends the results from [2,3,4] to the more general case of *S2T* model transformations.

6 Conclusion and Ongoing Work

We have considered the semantical correctness and completeness of model transformations based on simulation specifications (typed graph transformation systems). The main results show under which conditions an *S2T* model transformation is semantically correct and complete. The theory has been presented

in the DPO-approach for typed graphs, but it can also be extended to typed attributed graphs, where injective graph morphisms are replaced by suitable classes M and M' of typed attributed graph morphisms for rules and NACs, respectively [8]. The results have been used to analyze an *S2T* transformation of a production system (a domain-specific visual model) to Petri nets. We also discuss the requirement of a model transformation *S2T* to be *behavior-preserving* in the sense that for each target model in VL_T the simulations via rules in P_T correspond to simulations via rules in the target semantics, given by $P_{\bar{T}}$ (e.g. the Petri net firing rules) and vice versa. Work is in progress to establish formal criteria for semantically correct and complete *S2T* model transformations to be also behavior-preserving w.r.t. a given target language semantics.

Future work is planned to analyze in more detail our *I2T* completeness condition, to automatize our approach (e.g. check the correctness and completeness conditions automatically by a tool) and to apply the approach to triple graph grammars [12], nowadays widely used for model transformation specification.

References

1. Narayanan, A., Karsai, G.: Using Semantic Anchoring to Verify Behavior Preservation in Graph Transformations. In: Proc. Workshop on Graph and Model Transformation (GraMoT 2006). EC-EASST, EASST, vol. 4 (2006)
2. Ermel, C.: Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation. PhD thesis, Technische Universität Berlin,fak, IV, Books on Demand, Norderstedt (2006)
3. Ermel, C., Ehrig, H., Ehrig, K.: Semantical Correctness of Simulation-to-Animation Model and Rule Transformation. In: Proc. Workshop on Graph and Model Transformation (GraMoT 2006). EC-EASST, vol. 4, EASST (2006)
4. Ermel, C., Ehrig, H.: Behavior-preserving simulation-to-animation model and rule transformation. In: Proc. Workshop on Graph Transformation for Verification and Concurrency (GT-VC 2007). ENTCS, vol. 213(1), pp. 55–74. Elsevier Science, Amsterdam (2008)
5. Ehrig, H., Ermel, C.: Semantical Correctness and Completeness of Model Transformations using Graph and Rule Transformation: Long Version. Technical Report 2008-13, TU Berlin (2008), `http://iv.tu-berlin.de/TechnBerichte/2008/2008-13.pdf`
6. de Lara, J., Vangheluwe, H.: Translating Model Simulators to Analysis Models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 77–92. Springer, Heidelberg (2008)
7. Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed Graph Transformation with Node Type Inheritance. Theoretical Computer Science 376(3), 139–163 (2007)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science. Springer, Heidelberg (2006)
9. Kreowski, H.-J.: A Comparison between Petri Nets and Graph Grammars. In: Noltemeier, H. (ed.) WG 1980. LNCS, vol. 100, pp. 1–19. Springer, Heidelberg (1981)

10. Ehrig, H., Ehrig, K.: Overview of Formal Concepts for Model Transformations based on Typed Attributed Graph Transformation. In: Proc. Workshop on Graph and Model Transformation (GraMoT 2005). ENTCS, vol. 152. Elsevier, Amsterdam (2005)
11. Varró, D.: Automated formal verification of visual modeling languages by model checking. Software and System Modeling 3(2), 85–113 (2004)
12. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)