Semantical Correctness and Completeness of Model Transformations using Graph and Rule Transformation: Long Version

Hartmut Ehrig and Claudia Ermel
Technische Universität Berlin, Germany
{lieske,ehrig}@cs.tu-berlin.de

Forschungsberichte des Fachbereichs Informatik
Bericht-Nr. 2008/13, ISSN 1436-9915

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 result of this rule transformation is compared with 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 result is applied to analyze the behavior of a model transformation from a domain-specific visual language for production systems to Petri nets.
Keywords: graph transformation, visual languages, simulation, model transformation, rule transformation, semantical correctness, semantical completeness

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 [1] and Petri nets [2]. 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. 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. [3] 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 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 [4–6]. The result is applied to analyze the behavior of a model transformation from a domain-specific language for production systems to Petri nets.

This technical report is the long version of our paper presented at the International Conference on Graph Transformation 2008 [7], giving unabridged formal definitions, full proofs of all theorems and more details of the case study.

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 for correct and complete model transformations defined by graph rules are worked out. The main result is applied to our running example, showing that the model transformation from production systems to Petri nets is semantically correct and complete. Section 5 discusses related work, and Section 6 concludes the 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 [8]). 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. 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 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

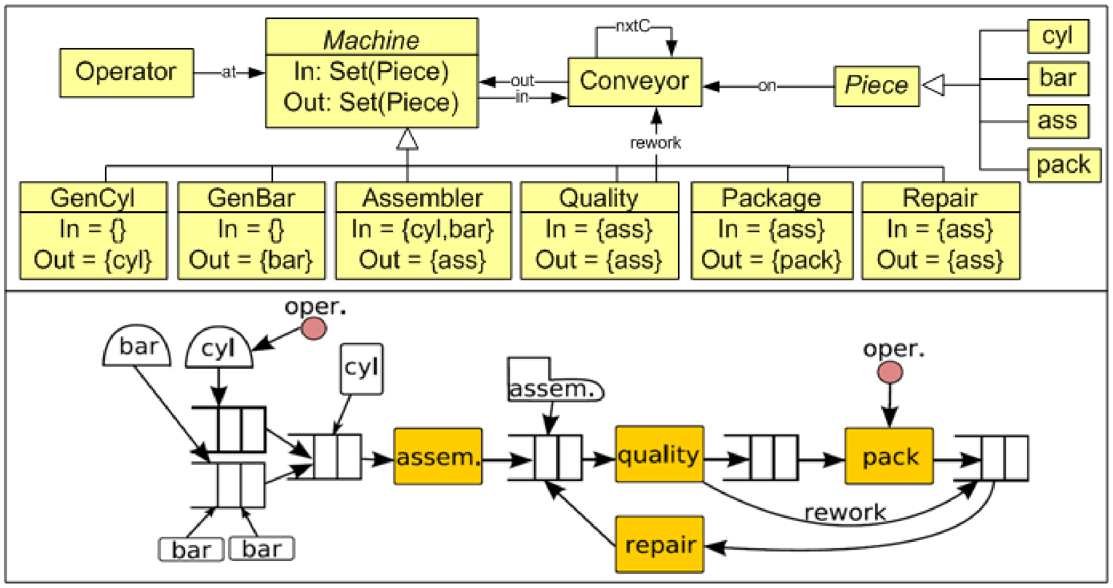


Fig. 1. Type Graph for Producer Systems and Instance Graph

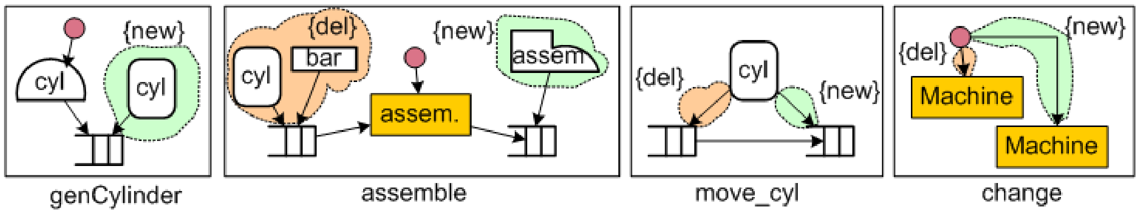


Fig. 2. Some Simulation Rules for Production Systems

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 [9]. 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) [10] which have proven to be an adequate formalism for visual language (VL) modeling. A VL is modeled by a type graph capturing the definition of 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)* [10], 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 the simulation alphabet TG_S, as a sublanguage of the modeling language VL, such 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 of the *S2I* transformation phase 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. Note that these two phases allow us to consider only non-deleting rules in the *S2I* transformation phase.

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*, 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. Note that a conveyor is transformed to four different places, thus realizing a flattening from our model with distinct piece types to a P/T net with indistinguishable tokens. 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*) and for each

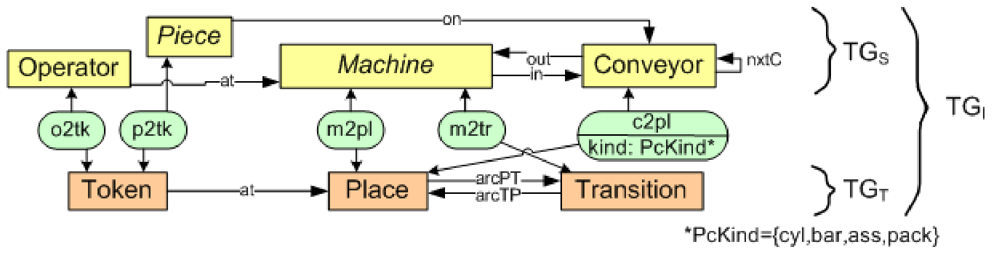


Fig. 3. Type Graph TG_I for the *ProdSystem2PetriNet* Model Transformation

machine which is connected to one or more conveyors (rules *first_in2tr*, *nxt_in2tr* and *first_out2tr*, *nxt_out2tr*). While rules *first_in2tr* and *first_out2tr* are applied only if there exists not yet a transition connected to a machine (modeled by suitable NACs which are not depicted), rules *nxt_in2tr* and *nxt_out2tr* are applied in the case that a machine is already connected to a transition and just add an arc connecting the existing transition to an existing conveyor place of the right kind. 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

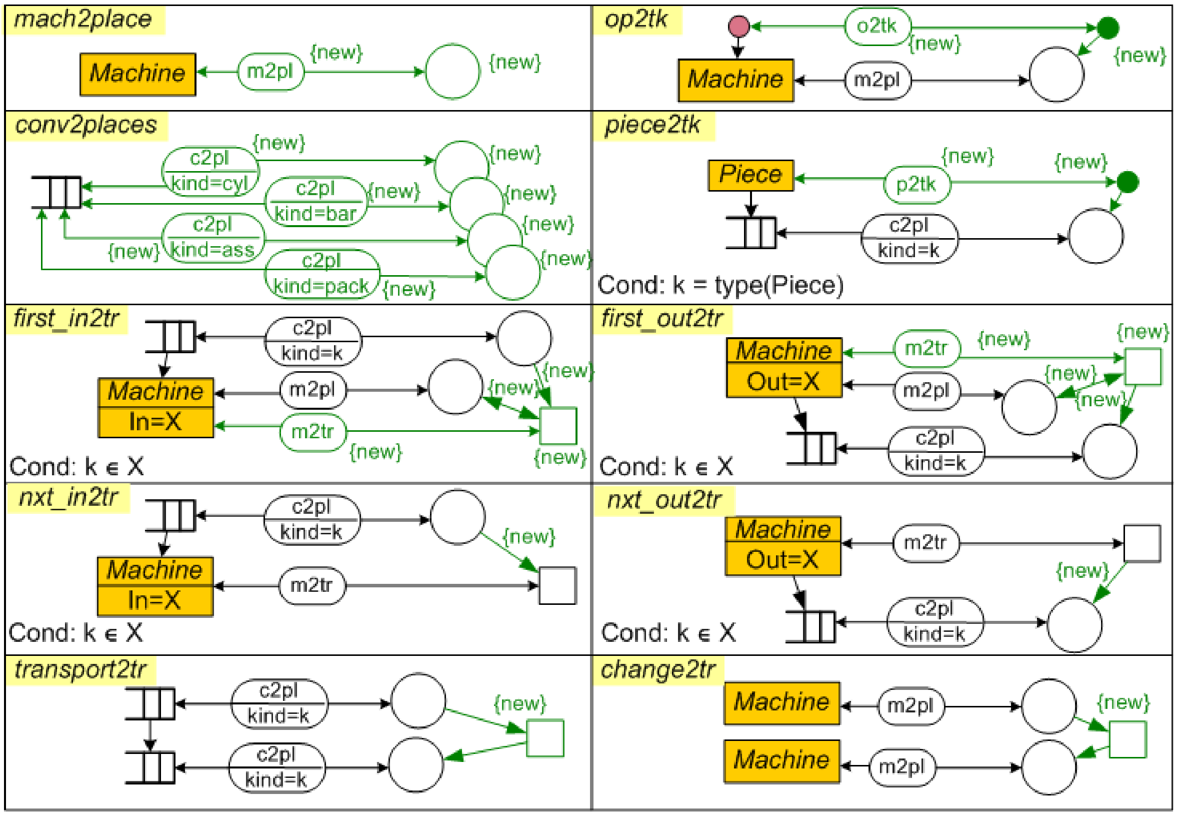


Fig. 4. *ProdSystem2PetriNet* Model Transformation Rules

S2I$_M$-transformation is illustrated in Fig. 5, where a part from the model shown in Fig. 1 has been transformed. The model transformation rules in Fig. 4 have been applied as long as possible, but at most once at the same match.

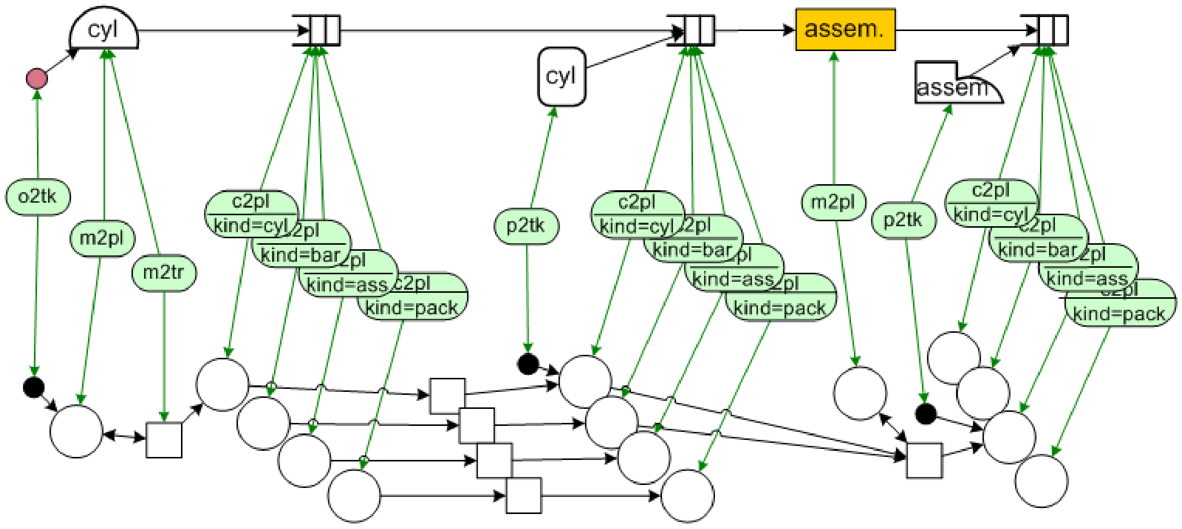


Fig. 5. *ProdSystem2PetriNet*: *S2I*$_M$ Model Transformation Result

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 Def. 3, we review a construction from [5,4], 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 *S2I* transformation rules are applied to each rule side of a simulation rule $p_S = (L_S \leftarrow K_S \rightarrow R_S)$ as long as possible, resulting in an integrated simulation rule $p_I = (L_I \leftarrow K_I \rightarrow R_I)$. Def. 3 defines rule transformation for the case without NACs. An extension to the case with NACs is given in [5,4].

Definition 3. (Transformation of Rules by Non-Deleting Rules) *Given a non-deleting rule* $q = (L_q \rightarrow R_q)$ *and a rule* $p_1 = (L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1)$*, then* q *is applicable to* p_1 *leading to a* rule transformation step $p_1 \overset{q}{\Longrightarrow} p_2$ *, if the precondition of one of the following three cases is satisfied, and* $p_2 = (L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2)$ *is defined according to the corresponding construction.*

$$\begin{tikzcd}
 & L_q \arrow[rr, "q"] \arrow[d, "h"'] \arrow[rrd, phantom, "(1)"] & & R_q \arrow[d] \\
 & K_1 \arrow[rr, "q_K"] \arrow[dl, "l_1"'] \arrow[dd, "r_1"] \arrow[d, phantom, "(1a)"] & & K_2 \arrow[dl, "l_2"] \arrow[dd, "r_2"] \\
L_1 \arrow[rr, "q_L"] & & L_2 & \\
 & R_1 \arrow[rr, "q_R"'] \arrow[rr, phantom, bend left, "(1b)"] & & R_2
\end{tikzcd}$$

Case (1)
Precondition (1)*: There is a match* $L_q \xrightarrow{h} K_1$.
Construction (1)*:* K_2*,* L_2*, and* R_2 *are defined by pushouts* $(1), (1a)$ *and* $(1b)$*, leading to injective morphisms* l_2 *and* r_2*.*

Case (2)

Precondition (2)*: There is no match $L_q \xrightarrow{h} K_1$, but a match $L_q \xrightarrow{h'} L_1$.*
Construction (2)*: L_2 is defined by pushout* (2)*, and $K_2 = K_1$, $R_2 = R_1$, $r_2 = r_1$, and $l_2 = q_L \circ l_1$.*

$$\begin{tikzcd} L_q \arrow[r, "q"] \arrow[d, "h'"'] \arrow[dr, phantom, "(2)"] & R_q \arrow[d] \\ L_1 \arrow[r, "q_L"] & L_2 \end{tikzcd}$$

Case (3)

Precondition (3)*: There are no matches $L_q \xrightarrow{h} K_1$ and $L_q \xrightarrow{h'} L_1$, but there is a match $L_q \xrightarrow{h''} R_1$.*
Construction (3)*: R_2 is defined by pushout* (3)*, and $L_2 = L_1$, $K_2 = K_1$, $l_2 = l_1$, and $r_2 = q_L \circ r_1$.*

$$\begin{tikzcd} L_q \arrow[r, "q"] \arrow[d, "h''"'] \arrow[dr, phantom, "(3)"] & R_q \arrow[d] \\ R_1 \arrow[r, "q_R"] & R_2 \end{tikzcd}$$

Case (1) in Def. 3 corresponds to the notion of rule rewriting in [11], adapted to non-deleting *S2I* transformation rules. In Case (2), the *S2I* transformation rule q is not applicable to the interface K_1, but to the left-hand side of a rule p_1, and in Case (3), q is not applicable to K_1, but to the right-hand side of p_1. Note that it is possible that both Case (2) and Case (3) can be true for different matches of q. Then, q is applied in a first step to L_1 according to (2), and in a second step to R_1 according to (3).

Based on Def. 3 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 4. (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 *$S2I_R : P_S \to P_I$ 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 Def. 3) are applied as long as possible. The* integrated simulation rules *P_I are defined by $P_I = \{p_I | \ \exists\ p_S \in P_S \wedge\ p_S \overset{Q\ !}{\Longrightarrow} p_I\ \}$. This means $p_S \overset{Q\ !}{\Longrightarrow} p_I$ implies $p_S \in P_S$ and $p_I \in P_I$.*

Definition 5. ***(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.

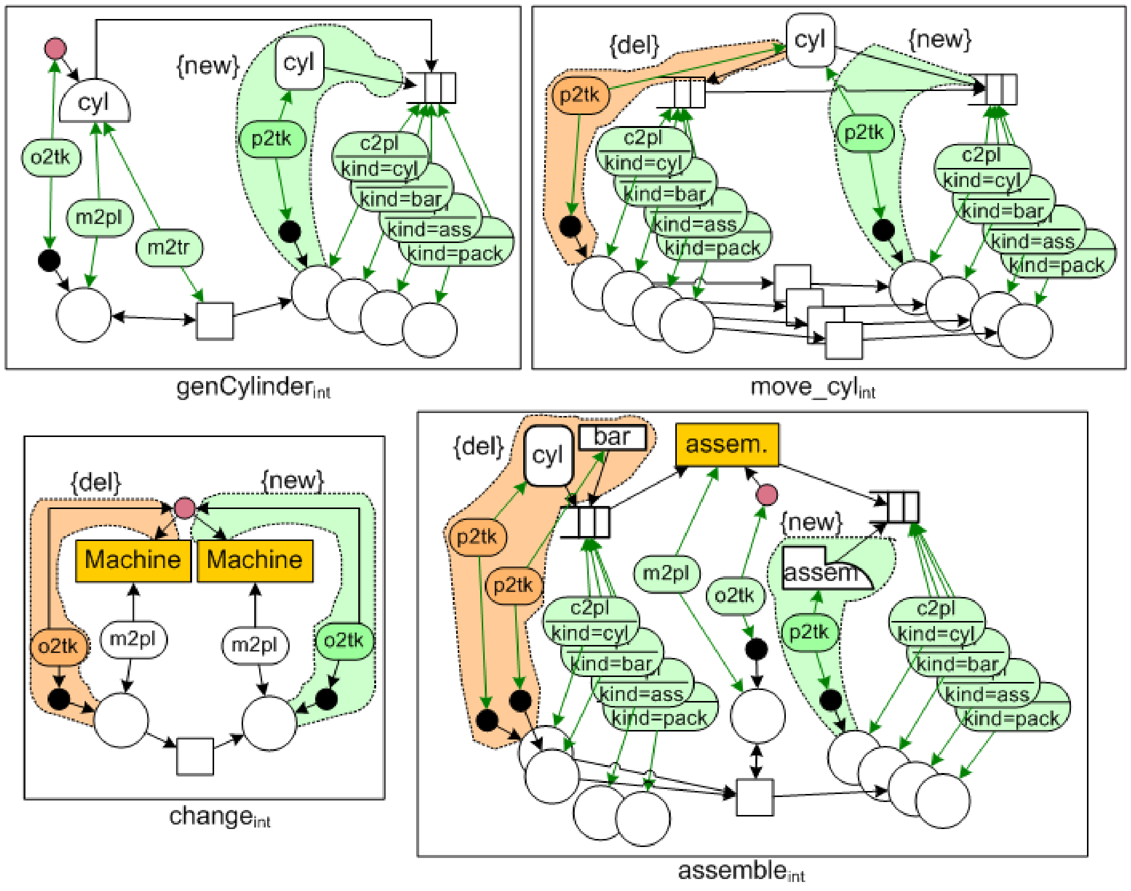


Fig. 6. Some Integrated Simulation Rules resulting from $S2I_R$ Transformation

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

1. $I2T$*:* $SimSpec_{VL_I} \rightarrow SimSpec_{VL_T}$*, called I2T* transformation*, is defined by* $I2T = (I2T_M : VL_I \rightarrow VL_T, I2T_R : P_I \rightarrow P_T)$ *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).

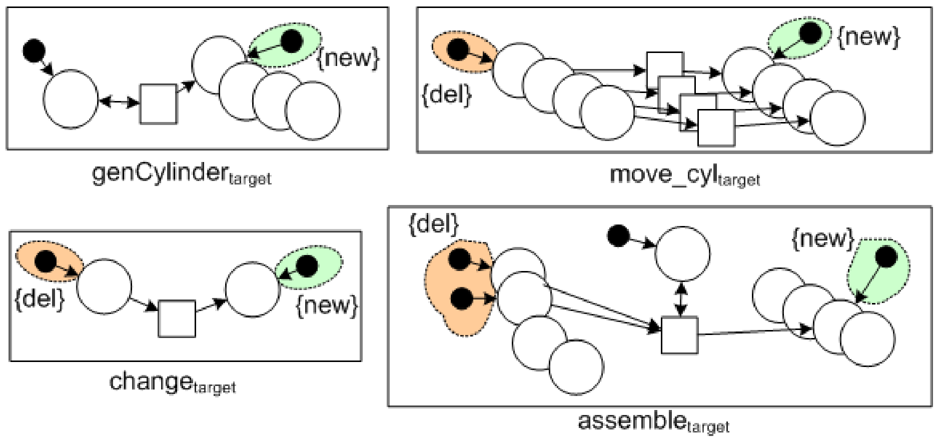


Fig. 7. Some Target Simulation Rules resulting from $I2T_R$ Transformation

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

Definition 7. ***(S2T Transformation)***
Given an S2I transformation $S2I : SimSpec_{VL_S} \rightarrow SimSpec_{VL_I}$*, and an I2T transformation* $I2T : SimSpec_{VL_I} \rightarrow SimSpec_{VL_T}$*, then* $S2T : SimSpec_{VL_S} \rightarrow SimSpec_{VL_T}$*, called S2T* transformation*, is defined by* $S2T = (S2T_M : VL_S \rightarrow VL_T, S2T_R : P_S \rightarrow P_T)$ *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

In this section, we continue the general theory of Section 3 and study behavior preservation, i.e. semantical correctness and completeness of model and rule transformations.

4.1 Semantical Correctness of *S2I* Transformations

In our case, semantical correctness of a $S2I$ 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 [5], it is shown that the following properties have to be fulfilled by an *S2I*-transformation in order to be semantically correct:

Definition 8. ***(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.* $H \overset{q}{\Longrightarrow} H'$*).*

In order to prove Theorem 1, we first show *local* semantical correctness in Proposition 1 where only one $S2I_M$-step (resp. $S2I_R$-step) is considered.

Proposition 1 (Local Semantical Correctness of *S2I*-Transformations).
Given an S2I-transformation $S2I : SimSpec_{VL_S} \to SimSpecVL_I$ *with* $S2I = (S2I_M : VL_S \to VL_I, S2I_R : P_S \to P_I)$ *and an* $S2I_R$*-transformation sequence* $p_S \overset{Q\ !}{\Longrightarrow} p_I$ *with intermediate* $S2I_R$*-step* $p_i \overset{q}{\Longrightarrow} p_{i+1}$ *with* $q \in Q$*. Then for each graph transformation step* $G_i \overset{p_i}{\Longrightarrow} H_i$ *with* $G_i, H_i \in \mathbf{Graphs_{TG_I}}$ *we have*

1. *Graph transformation steps* $G_i \overset{q_i}{\Longrightarrow} G_{i+1}$ *in Cases (1) and (2),* $G_i \overset{id}{\Longrightarrow} G_{i+1}$ *in Case (3),* $H_i \overset{q}{\Longrightarrow} H_{i+1}$ *in Cases (1) and (3), and* $H_i \overset{id}{\Longrightarrow} H_{i+1}$ *in Case (2) of Def. 3.*
2. *Graph transformation step* $G_{i+1} \overset{p_{i+1}}{\Longrightarrow} H_{i+1}$ *with* G_{i+1}*,* $H_{i+1} \in \mathbf{Graphs_{TG_I}}$

$$\begin{tikzcd}
G_i \arrow[rr, Rightarrow, "q\ /\ id"] \arrow[dd, Rightarrow] & & G_{i+1} \arrow[dd, Rightarrow] \\
 & p_i \arrow[r, Rightarrow, "q"] & p_{i+1} \\
H_i \arrow[rr, Rightarrow, "q\ /\ id"] & & H_{i+1}
\end{tikzcd}$$

Proof. We consider the respective pushout diagrams for $p_i \overset{q}{\Longrightarrow} p_{i+1}$ according to the three rule transformation cases in Def. 3, and show by pushout composition/decomposition that in each case we obtain the commuting double cube below where the two back squares comprise the given DPO for the transformation step $G_i \overset{p_i}{\Longrightarrow} H_i$, and in the front squares we get the required DPO for the transformation step $G_{i+1} \overset{p_{i+1}}{\Longrightarrow} H_{i+1}$.

$$\begin{tikzcd}
 & L_i \arrow[dl] \arrow[dd, "m_i"] & & K_i \arrow[ll, "l_i"'] \arrow[rr, "r_i"] \arrow[dl] \arrow[dd] & & R_i \arrow[dl] \arrow[dd] \\
L_{i+1} \arrow[dd, "m_{i+1}"'] & & K_{i+1} \arrow[ll] \arrow[rr] \arrow[dd] & & R_{i+1} \arrow[dd] & \\
 & G_i \arrow[dl] & & C_i \arrow[ll] \arrow[rr] \arrow[dl] & & H_i \arrow[dl] \\
G_{i+1} & & C_{i+1} \arrow[ll] \arrow[rr] & & H_{i+1} &
\end{tikzcd}
\qquad
\begin{tikzcd}
L_q \arrow[r, "q"] \arrow[d, "h"'] \arrow[dr, phantom, "(PO_I)"] & R_q \arrow[d] \\
K_i \arrow[r, "q_{i+1}"] & K_{i+1}
\end{tikzcd}$$

In Case (1) of Def. 3, we obtain the top squares as pushouts and then construct G_{i+1}, C_{i+1}, H_{i+1} as pushouts in the diagonal squares, leading to unique induced morphisms $C_{i+1} \to G_{i+1}$ and $C_{i+1} \to H_{i+1}$ such that the double cube commutes. By pushout composition/decomposition also the front and the bottom squares are pushouts. Furthermore, we obtain pushouts for the transformation steps $G_i \overset{q}{\Longrightarrow} G_{i+1}$ and $H_i \overset{q}{\Longrightarrow} H_{i+1}$ by composing pushout (PO_I) with the respective pushouts from the double cube. Cases (2) and (3) are handled similarly, with the difference that some morphisms in the respective double cubes are identities.

Based on the notions of termination and rule compatibility in Def. 8, we now extend local semantical correctness of *S2I* to semantical correctness. Note that the proof of Theorem 1 corresponds to the proof of Semantical Correctness of *S2A* in [5].

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. 4):*

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

$$\begin{tikzcd}
G_S \arrow[rr, Rightarrow, "Q\ \ !"] \arrow[dd, Rightarrow] & & G_I \arrow[dd, Rightarrow] \\
p_S \arrow[rr, Rightarrow, "Q\ \ !"] & & p_I \\
H_S \arrow[rr, Rightarrow, "Q\ \ !"] & & H_I
\end{tikzcd}$$

Proof. Given $S2I = (S2I_M : VL_S \to VL_I, S2I_R : P_S \to P_I)$ with terminating $S2I_M$, a simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ with $G_S \in VL_S$, and an $S2I_R$ transformation sequence $p_S \overset{Q\ !}{\Longrightarrow} p_I$ with $p_S = p_0 \overset{q_0}{\Longrightarrow} p_1 \overset{q_1}{\Longrightarrow} .. \overset{q_{n-1}}{\Longrightarrow} p_n = p_I$ with $n \geq 1$, then we can apply the Local Semantical Correctness Theorem 1 for $i = 0, .., n-1$, leading to the diagram below, which includes the case $n = 0$ with $G_S = G_0, H_S = H_0$ and $p_S = p_0 = p_I$, where no $q \in Q$ can be applied to p_S.

$$\begin{tikzcd}
G_S = G_0 \arrow[r, Rightarrow, "q_0"] \arrow[dd, Rightarrow, "p_S = p_0"] & G_1 \arrow[r, Rightarrow, "q_1"] \arrow[dd, Rightarrow, "p_1"] & G_2 \arrow[r, Rightarrow, "q_2"] \arrow[dd, Rightarrow, "p_2"] & \cdots \arrow[r, Rightarrow] & G_{n-1} \arrow[r, Rightarrow, "q_{n-1}"] \arrow[dd, Rightarrow, "p_{n-1}"] & G_n \arrow[dd, Rightarrow, "p_n = p_I"] \\
p_S = p_0 \arrow[rrrrr, Rightarrow, "Q!"] & & & & & p_n = p_I \\
H_S = H_0 \arrow[r, Rightarrow, "q_0"'] & H_1 \arrow[r, Rightarrow, "q_1"'] & H_2 \arrow[r, Rightarrow, "q_2"'] & \cdots \arrow[r, Rightarrow] & H_{n-1} \arrow[r, Rightarrow, "q_{n-1}"'] & H_n
\end{tikzcd}$$

If no $q \in Q$ can be applied to G_n and H_n anymore, we are ready, because the top sequence is $G_S \overset{Q\ !}{\Longrightarrow} G_n = G_I$, and the bottom sequence is $H_S \overset{Q\ !}{\Longrightarrow} H_n = H_I$.

Now assume that we have $q_n \in Q$ which is applicable to G_n leading to $G_n \overset{q_n}{\Longrightarrow} G_{n+1}$. Then, rule compatibility implies parallel independence with $G_I \overset{p_I}{\Longrightarrow} H_I$, and the Local Church Rosser Theorem [10] leads to square (n):

$$\begin{tikzcd}
G_n \arrow[r, Rightarrow, "q_n"] \arrow[d, Rightarrow, "p_I"] \arrow[dr, phantom, "(n)"] & G_{n+1} \arrow[r, Rightarrow] \arrow[d, Rightarrow, "p_I"] & \cdots \arrow[r, Rightarrow] & G_{m-1} \arrow[r, Rightarrow, "q_{m-1}"'] \arrow[d, Rightarrow, "p_I"] & G_m = G_I \arrow[d, Rightarrow, "p_I"] \\
H_n \arrow[r, Rightarrow, "q_n"] & H_{n+1} \arrow[r, Rightarrow] & \cdots \arrow[r, Rightarrow] & H_{m-1} \arrow[r, Rightarrow, "q_{m-1}"] & H_m = H_I
\end{tikzcd}$$

This procedure can be repeated as long as rules $q_i \in Q$ are applicable to G_i for $i \geq n$. Since *S2I*$_M$ is terminating, we have some $m > n$ such that no $q \in Q$ is applicable to G_m anymore, leading to a sequence $G_S = G_0 \overset{Q\ !}{\Longrightarrow} G_m = G_I$. Now assume that there is some $q \in Q$ which is still applicable to H_m leading to $H_m \overset{q}{\Longrightarrow} H_{m+1}$. Now rule compatibility implies sequential independence of $G_m \overset{p_I}{\Longrightarrow} H_m \overset{q}{\Longrightarrow} H_{m+1}$. In this case, the Local Church Rosser Theorem would lead to a sequence $G_m \overset{q}{\Longrightarrow} G_{m+1} \overset{p_I}{\Longrightarrow} H_{m+1}$ which contradicts the fact that no $q \in Q$ is applicable to G_m anymore. This implies that also $H_0 \overset{Q\ *}{\Longrightarrow} H_n \overset{Q\ *}{\Longrightarrow} H_m$ is terminating, leading to the required sequence $H_S = H_0 \overset{Q\ !}{\Longrightarrow} H_m = H_I$. As long as rules $q_i \in Q$ are applicable to G_i for $i \geq n$, the termination of the *S2I*$_M$ transformation ensures that we have some $m > n$ such that no $q \in Q$ is applicable to G_m anymore, leading to a sequence $G_S = G_0 \overset{Q\ !}{\Longrightarrow} G_m = G_I$. Furthermore, the rule compatibility of *S2I* ensures that whenever $q_i \in Q$ is applicable to G_i for $i \geq n$, then q_i is also applicable to H_i (and vice versa), which implies squares $(n+1), ..(m-1)$ with $\overset{p_I}{\Longrightarrow}$ in the vertical direction, and we get the required sequences $G_S = G_0 \overset{Q\ !}{\Longrightarrow} G_m = G_I$ and $H_S = H_0 \overset{Q\ !}{\Longrightarrow} H_m = H_I$.

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 for *ProdSystem2PetriNet*, 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 $G_I : L_q \xrightarrow{h} G_I \xleftarrow{m} L_I$. We find that each overlap either is preserved by both rules, or that $h(L_q)$ 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, and 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. 6 as the restriction of the integrated model graph and the integrated simulation rules to the type graph TG_T of the target VL.

For the proof of Theorem 2 we need a property of a type graph embedding, defined by an injective type graph morphism $f_{TG} : TG_1 \to TG_2$. Type graph embeddings induce TGTS embeddings (morphisms between type graph transformation systems) [4] which correspond to the notion of restriction of graph transformation systems, i.e. for a type graph embedding $f_{TG} : TG_1 \to TG_2$, a TGTS embedding exists between two typed graph transformation systems $TGTS_1 = (TG_1, P_1)$ and $TGTS_2 = (TG_2, P_2)$ if $P_1 = P_2|_{TG_1}$).

Proposition 2 (TGTS Embeddings reflect the Behavior). *Given a type graph embedding, i.e. an injective type graph morphism $f_{TG} : TG_1 \to TG_2$ and a TGTS embedding from $TGTS_1 = (TG_1, P_1)$ to $TGTS_2 = (TG_2, P_2)$. Then, the TGTS embedding reflects the behavior in the sense that if we have a transformation $G_2 \overset{p_2,m_2}{\Longrightarrow} H_2$ in $TGTS_2$, we get the transformation $G_1 \overset{p_1,m_1}{\Longrightarrow} H_1$ in $TGTS_1$, where G_1, H_1, p_1 and m_1 are restrictions of G_2, H_2, p_2 and m_2 to TG_1, respectively.*

Proof. Basically, the proof works by construction of the double cube shown below, where the front squares are pushouts corresponding to a rewriting step $G_2 \overset{p_2}{\Longrightarrow} H_2$ in the DPO approach, applying the rule $p_2 = (L_2 \leftarrow I_2 \to R_2)$ to graph G_2. We can contruct L_1, K_1 and R_1 as restrictions of L_2, K_2 and R_2, respectively, and G_1, D_1 and H_1 as restrictions of G_2, D_2 and H_2, such that the diagonal, bottom and top squares are pullbacks and the double cube commutes. Thus, the Van-Kampen property (see [10]) can be used to prove that the back squares are also pushouts, which correspond to the rewriting step $G_1 \overset{p_1}{\Longrightarrow} H_1$ in the DPO approach, applying the rule $p_1 = (L_1 \leftarrow K_1 \to R_1)$ to G_1.

$$\begin{tikzcd}
 & & & L_1 \arrow[dl] \arrow[dd] & & K_1 \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & R_1 \arrow[dl] \arrow[dd] & & \\
 & & L_2 \arrow[dd] & & I_2 \arrow[ll] \arrow[rr] \arrow[dd] & & R_2 \arrow[dd] & & & \\
 & TG_1 \arrow[dl] & & G_1 \arrow[ll] \arrow[dl] & & D_1 \arrow[ll] \arrow[rr] \arrow[dl] & & H_1 \arrow[rr] \arrow[dl] & & TG_1 \arrow[dl] \\
TG_2 & & G_2 \arrow[ll] & & D_2 \arrow[ll] \arrow[rr] & & H_2 \arrow[rr] & & TG_2 &
\end{tikzcd}$$

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. 6. 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 $G_T \overset{p_T}{\Longrightarrow} H_T$*

$$\begin{tikzcd}
G_I \arrow[r, "I2T_M"] \arrow[dd, Rightarrow] & G_T \arrow[dd, Rightarrow] \\
p_I \arrow[r, "I2T_R"] & p_T \\
H_I \arrow[r, "I2T_M"] & H_T
\end{tikzcd}$$

Proof. The semantical correctness of *I2T* transformations holds because due to the definition of *I2T* as restriction of the integrated model G_I to G_T and of the integrated rules p_I to p_T, we have a TGTS embedding from $SimSpec_{VL_T}$ to $SimSpec_{VL_I}$. TGTS embeddings reflect the behavior according to Proposition 2. Hence, if we have a transformation $G_I \overset{p_I,m_I}{\Longrightarrow} H_I$ in $SimSpec_{VL_I}$, we get the transformation $G_T \overset{p_T,m_T}{\Longrightarrow} H_T$

in $SimSpec_{VL_T}$, where G_T, H_T, p_T and m_T are restrictions of G_I, H_I, p_I and m_I, respectively.

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 9. *(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*$_M(p_I) = p_S$.

We call a model transformation rule $L \xrightarrow{q} R$ *faithful* if its restriction $q|_{TG_S}$ to the source language is the identity. For the proof of Theorem 3, we first need to show that an *S2I* transformation is faithful if all rules $q \in Q$ are faithful (see Prop. 3).

Definition 10 (Faithful Model Transformation Rule). *A nondeleting model transformation rule $q : L \to R \in Q$ is called* faithful*, if the restriction of q to TG_S is the identic rule: $q|_{TG_S} = id$.*

Proposition 3 (Faithful *S2I* **Transformation).** *Given Q as set of faithful model transformation rules $q : L \to R$. Then, each $G \overset{q}{\Longrightarrow} H$ with $f : G \to H$ can be chosen such that $f|_{TG_S} = id$.*

$$\begin{tikzcd}
 & L' \arrow[rr] \arrow[dl] \arrow[dd, tail, "q'"] & & L \arrow[dl] \arrow[dd, tail, "q"] \\
G' \arrow[rr, tail] \arrow[dd, tail, "f'"] & & G \arrow[dd, tail, "f"] & \\
 & R' \arrow[rr, dashed] \arrow[dl] \arrow[ddl] & & R \arrow[dl] \arrow[ddl] \\
H' \arrow[rr, tail] \arrow[d] & & H \arrow[d] & \\
TG_S \arrow[rr, tail] & & TG_I &
\end{tikzcd}$$

Proof. Given $G \overset{q}{\Longrightarrow} H$ by the right pushout, we construct L', R', G' and H' as restrictions of L, R, G and H, respectively, such that all squares in the cube are pullbacks. Then, the van Kampen property implies that the left square is a pushout where q faithful implies $q' = id$ and hence, w.l.o.g. also $f' = id$.

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 $p_I \in P_I$:*

$$\begin{tikzcd}
G_I \arrow[r, "I2S_M"] \arrow[dd, Rightarrow] & G_S \arrow[dd, Rightarrow] \\
p_I \arrow[r, "I2S_R"] & p_S \\
H_I \arrow[r, "I2S_M"] & H_S
\end{tikzcd}$$

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. The semantical completeness of *S2I* holds due to the fact that the *I2S* backward transformation induces a typed graph transformation system (TGTS) embedding from $SimSpec_{VL_S}$ to $SimSpec_{VL_I}$ (see [6]). TGTS embeddings reflect the behavior according to Prop. 2. Hence, if we have a transformation $G_I \overset{p_I,m_I}{\Longrightarrow} H_I$ in $SimSpec_{VL_I}$, we get the transformation $G_S \overset{p_S,m_S}{\Longrightarrow} H_S$ in $SimSpec_{VL_S}$ with $I2S_M(G_I) = G_S$, $I2S_M(H_I) = H_S$ and $I2S_R(p_I) = p_S$. It remains to show that $S2I_M(G_S) = G_I$ (and similarly $S2I_M(H_S) = H_I$ and $S2I_R(p_S) = p_I$). In fact, $G_I \in VL_I$ implies existence of $G'_S \in VL_S$ with $S2I_M(G'_S) = G_I$, and by *S2I* faithful we have $I2S_M(G_I) = G'_S$. This implies $G_S = G'_S$ and $S2I_M(G_S) = G_I$.

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 11 (*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

$$\begin{tikzcd}
 & L_T \arrow[dl] \arrow[dd] & & K_T \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & R_T \arrow[dl] \arrow[dd] \\
L_I \arrow[dd] & & K_I \arrow[ll] \arrow[rr] \arrow[dd] & & R_I \arrow[dd] & \\
 & G_T & & D_T \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & H_T \\
G_I & & D_I \arrow[ll] \arrow[rr] \arrow[dd] & & H_I & \\
 & & & TG_T \arrow[dl] & & \\
 & & TG_I & & &
\end{tikzcd}$$

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 $G_I, H_I \in VL_I$.*

For the proof of Theorem 4, we need an additional proposition which ensures the existence of a pullback in a double cube under certain conditions:

Proposition 4 (Existence of Pullback in Double Cube). *Given the commutative double cube to the right, where we have pushouts in the upper back and the upper front squares, and pullbacks in the composite left, the upper right, upper top, upper bottom, upper left and lower right squares. Moreover, all horizontal morphisms are injective. Then, the lower left square is a pullback.*

$$\begin{tikzcd}
& L_T \arrow[dl] \arrow[dd] & & K_T \arrow[ll] \arrow[dl] \arrow[dd] \\
L_I \arrow[dd] & & K_I \arrow[ll] \arrow[dd] & \\
& G_T \arrow[dl] \arrow[dd] & & D_T \arrow[ll] \arrow[dl] \arrow[dd] \\
G_I \arrow[dd] & & D_I \arrow[ll] \arrow[dd] & \\
& TG_T \arrow[dl] & & TG_T \arrow[ll, "id"'] \arrow[dl] \\
TG_I & & TG_I \arrow[ll, "id"] &
\end{tikzcd}$$

Proof.

– **Part 1** *(Existence)*: Given $x_1 \in TG_T, x_2 \in G_I$ with $x_1 \mapsto x_0 \mapsfrom x_2$ for $x_0 \in TG_I$, we show that $\exists z_1 \in G_T$ s.t. $z_1 \mapsto x_1$ and $z_1 \mapsto x_2$.

Since G_I is pushout object, we have two cases:
- **Case 1**: $\exists y_2 \in L_I : y_2 \mapsto x_2$
  (PB in composite left square) $\implies \exists y_1 \in L_T : y_1 \mapsto y_2, y_1 \mapsto x_1$. Let $z_1 = (L_T \to G_T)(y_1) \implies z_1 \in G_T$ with $z_1 \mapsto x_1$ and $(G_T \to G_I)(z_1) = (L_T \to G_T \to G_I)(y_1) = (L_T \to L_I \to G_I)(y_1) = x_2$
- **Case 2**: $\exists y_2' \in D_I : y_2' \mapsto x_2$
  (PB in lower right square) $\implies \exists y_1' \in D_T : y_1' \mapsto y_2', y_1' \mapsto x_1$. Let $z_1 = (D_T \to G_T)(y_1') \implies z_1 \in G_T$ with $z_1 \mapsto x_1$ and $(G_T \to G_I)(z_1) = (D_T \to G_T \to G_I)(y_1') = (D_T \to D_I \to G_I)(y_1') = x_2$

– **Part 2** *(Uniqueness)*: Given $x_1, x_2 \in G_T$ with
$(G_T \to G_I)(x_1) = x_0 = (G_T \to G_I)(x_2)$, and
$(G_T \to TG_T)(x_1) = (G_T \to TG_T)(x_2)$, we have to show that $x_1 = x_2$.

Due to G_I being pushout object, we have three cases:
- **Case 1**: $\exists y_1 \in L_T, y_2 \in D_T, (L_T \to G_T)(y_1) = x_1, (D_T \to G_T(y_2) = x_2$
  Let $z_1 = (L_T \to L_I)(y_1), z_2 = (D_T \to D_I)(y_2)$. Then, $z_1 \mapsto x_0, z_2 \mapsto x_0$. (PB in upper front square) $\implies \exists z_0 \in K_I : z_0 \mapsto z_1, z_0 \mapsto z_2$.
  (PB in upper right square) $\implies \exists z_0' \in K_T : z_0' \mapsto z_0, z_0' \mapsto y_2$.
  $(L_T \to L_I)$ injective $\implies z_0' \mapsto y_1$
  (upper back square is commutative) $\implies x_1 = x_2$.
- **Case 2**: $\exists y_1, y_2 \in D_T, (D_T \to G_T)(y_1) = x_1, (D_T \to G_T)(y_2) = x_2$
  Let $z_1 = (D_T \to D_I)(y_1)$ and $z_2 = (D_T \to D_I)(y_2) \implies (D_I \to G_I)(z_1) = (D_I \to G_I)(z_1) = x_0$
  $(D_I \to G_I)$ injective $\implies z_1 = z_2$
  $(D_T \to D_I)$ injective $\implies y_1 = y_2 \implies x_1 = x_2$

- **Case 3**: $\exists y_1, y_2 \in L_T, (L_T \to G_T)(y_1) = x_1, (L_T \to G_T)(y_2) = x_2$
  Let $z_0 = (G_T \to TG_T)(x_1) = (G_T \to TG_T)(x_2) \implies (z_0' = (TG_T \to TG_I)(z_0) = (G_I \to TG_I)(x_0)$
  Let $z_1 = (L_T \to L_I)(y_1)$ and $z_2 = (L_T \to L_I)(y_2) \implies (L_I \to G_I)(z_1) = (L_I \to G_I)(z_2) = x_0$
  (PB in upper left square) $\implies$
  $\exists y_{12} \in L_T : (L_T \to L_I)(y_{12}) = z_1, (L_T \to G_T)(y_{12}) = x_2$
  $\exists y_{21} \in L_T : (L_T \to L_I)(y_{21}) = z_2, (L_T \to G_T)(y_{21}) = x_1$
  (PB in left composite square) $\implies \exists! y \in L_T$:
  $(L_T \to L_I)(y) = z_1$, and $(L_T \to TG_T)(y) = z_0$
  $\implies y_{12} = y_1 \implies x_1 = (L_T \to G_T)(y_1) = (L_T \to G_T)(y_{12}) = x_2$

Theorem 4 (Semantical Completeness of *I2T*).
Each I2T transformation $I2T = (I2T_M, I2T_R)$ which satisfies the I2T completeness condition (see Def. 11) 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

$$\begin{tikzcd}
G_I \arrow[r, "I2T_M"'] \arrow[dd, Rightarrow] & G_T \arrow[dd, Rightarrow] \\
p_I \arrow[r, "I2T_R"] & p_T \\
H_I \arrow[r, "I2T_M"] & H_T
\end{tikzcd}$$

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

Proof. Given a target simulation rule p_T which is restriction of an integrated simulation rule p_I to TG_T, i.e. in the diagram to the right, the top left and top right squares are pullbacks. Given a target transformation $G_T \overset{p_T}{\Longrightarrow} H_T$, i.e. the upper back squares are pushouts. Since we have $G_T \in VL_T$, and the completeness condition (part 1) is satisfied, we have a context graph D_I typed over TG_I and two morphisms $K_I \to D_I$ and $K_T \to D_T$ such that the diagonal squares are pullbacks. We construct the graphs G_I and H_I as pushout objects, e.g. we get two pushouts in the upper front squares (the DPO for the transformation $G_I \overset{p_I}{\Longrightarrow} H_I$) with $G_S, H_S \in VL_I$ by completeness condition (part 2). We also get the morphisms $G_I \to TG_I$ and $H_I \to TG_I$ as unique pushout morphisms.

$$\begin{tikzcd}
 & L_T \arrow[dl] \arrow[dd] & & K_T \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & R_T \arrow[dl] \arrow[dd] \\
L_I \arrow[dd] & & K_I \arrow[ll] \arrow[rr] \arrow[dd] & & R_I \arrow[dd] & \\
 & G_T \arrow[dl] \arrow[dd] & & D_T \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & H_T \arrow[dl] \arrow[dd] \\
G_I \arrow[dd] & & D_I \arrow[ll] \arrow[rr] \arrow[dd] & & H_I \arrow[dd] & \\
 & TG_T \arrow[dl] & & TG_T \arrow[ll, "id"'] \arrow[rr, "id"] \arrow[dl] & & TG_T \arrow[dl] \\
TG_I & & TG_I \arrow[ll, "id"] \arrow[rr, "id"'] & & TG_I &
\end{tikzcd}$$

Moreover, we get the morphisms $G_T \to G_I$ and $H_T \to H_I$ as unique pushout morphisms from the DPO pushouts in the upper back squares, such that the upper left square, the bottom squares of the upper cubes, and the upper right square commute. Using the Van-Kampen property, we get pullbacks in the upper left and right squares and in the upper bottom squares. Now we have the situation that we can apply Proposition 4 to both the left and the right double cube, and hence we get that the lower left square and the lower right square are pullbacks. This implies $G_T = G_I|_{TG_T}$ and $H_T = H_I|_{TG_T}$.

Example 7. We show that our *ProdSystem2PetriNet I2T* transformation up to now does not fulfill the completeness condition and discuss an adaption of the model transformation rules in order to achieve semantical completeness of the *I2T* transformation. 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 target rules $move\_cyl_{target}$ and $change_{target}$ in Fig. 7. Both rules are applicable to a target graph $G_T \in VL_T$ if there exists a match from the "biggest" rule $move\_cyl_{target}$ to G_T. Thus, when applying either rule $move\_cyl_{target}$ or rule $change_{target}$ to G_T, we 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 $K_I \rightarrow D_I$ such that the pushout objects G_I and H_I are *not* in VL_I. In Fig. 8, such a situation is shown where the target rule was $change_{target}$ was constructed by restricting the integrated rule $change_{int}$ to the Petri net target language.

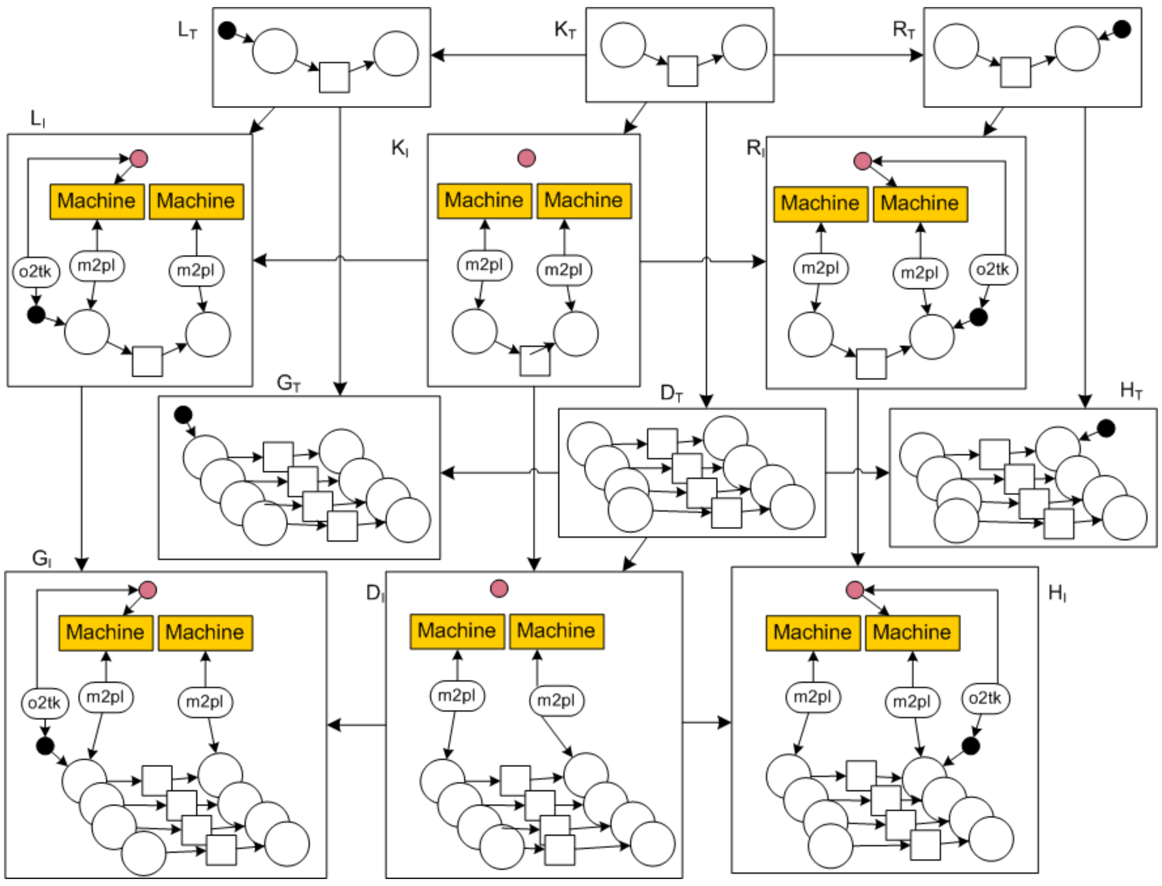


Fig. 8. Violated Completeness Condition in *ProdSystem2PetriNet*

But rule $change_{target}$ is applied to an occurrence in the host graph (only a part of this host graph G_T is shown in Fig. 8), where the places did not originally correspond to machines but to conveyor belts. Thus, when the pushout objects G_I and H_I are constructed, we get graphs which do not belong to VL_I, i.e. it is not possible to derive these graphs by *S2I* transformation from any valid source production system (again, only a part of G_I and H_I are shown in Fig. 8).

This might happen because our model transformation "forgets" information, i.e. when looking at a target rule (typed over the Petri net language), we do not know anymore, from which integrated rule this target rule was constructed. In order to avoid such situations, 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 which keep the information about the original role of the element. For example, by extending the model transformation rules, we annotate each place originating from a machine by the type of machine (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. In this case, 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. 8) and S2I faithful, and* $I2T : SimSpec_{VL_I} \to SimSpec_{VL_T}$*, with I2T satisfying the completeness condition (Def. 11), 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* $p_S \overset{Q\,!}{\Longrightarrow} p_I \xrightarrow{|_{TG_T}} p_T$ *we have*

1. $S2T_M$*-trafo* $S2T_M(G_S) = G_T : G_S \overset{Q!}{\Longrightarrow} G_I \xrightarrow{|_{TG_T}} G_T$,
   $S2T_M$*-trafo* $S2T_M(H_S) = H_T : H_S \overset{Q!}{\Longrightarrow} H_I \xrightarrow{|_{TG_T}} H_T$, *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*

$$\begin{tikzcd}
G_S \arrow[r, Rightarrow, "S2I_M"] \arrow[rr, bend left, "S2T_M"] \arrow[dd, Rightarrow] & G_I \arrow[r, "I2T_M"] \arrow[dd, Rightarrow] & G_T \arrow[dd, Rightarrow] \\
p_S \arrow[r, Rightarrow, "S2I_R"] \arrow[rr, bend left, "S2T_R"] & p_I \arrow[r, "I2T_R"] & p_T \\
H_S \arrow[r, Rightarrow, "S2IM"] \arrow[rr, bend left, "S2T_M"] & H_I \arrow[r, "I2T_M"] & H_T
\end{tikzcd}$$

– $p_T = S2T_R(p_S)$,
– $G_T = S2T_M(G_S)$ *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* $G_S \overset{p_S}{\Longrightarrow} H_S$.

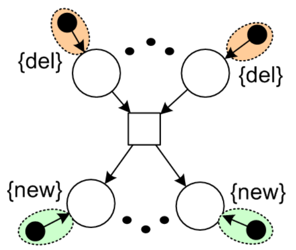
Proof. By semantical correctness of *S2I* and *I2T* (Theorems 1 and 2), we get directly the semantical correctness of $S2T = I2T \circ S2I$. By semantical completeness of *S2I* and *I2T* (Theorems 3 and 4), we get directly the semantical completeness of $S2T = I2T \circ S2I$.

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 [12]. 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 the rules with annotations. In our example, we find that all target rules without annotations which are applicable to G_T at matches which overlap in the activated transition and its environment, have the same transformation span, i.e. $G_T \leftarrow D_T \rightarrow H_T$ (they are semantically equivalent). This means for instance that the target rules in Fig. 7 are all semantically equivalent for a match from the "biggest" rule $move\_cyl_{target}$ to G_T, since they differ only in the context which is preserved in each rule. It can be checked easily that we have a similar situation for all other target rules. The NACs prevent that the target rules without annotations become applicable to transitions with a larger environment. With respect to the annotated target rules, all semantically equivalent target rules without annotations which are applicable at matches containing the same activated transition, correspond to exactly one application of an annotated target rule at this match, (so this annotated rule is semantically equivalent to the rules without annotations). Thus, we can omit the annotations in the target rules without effecting 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 never changed by the rules in P_T. Then the effect of applying rule p_T corresponds exactly to the effect of applying the rule for the corresponding transition type from $P_{\bar{T}}$. In our running example, 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* of model transformations (see [13] for an overview).

To ensure the semantical correctness of model transformations, Varró et al. [14] use graph transformation to specify the dynamic behavior of systems and generate a transition system for each model. Based on the transition system, a model checker verifies certain dynamic consistency properties by model checking the source and target models. In [3], a method is presented to verify the semantical equivalence for particular model transformations. It is shown by finding bisimulations that a target model preserves the semantics of the source model with respect to a particular property. This technique does not prove the correctness of the model transformation rules in general, as we propose in this paper. In [4–6], we consider *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. The animation specification A in [4–6] corresponds to an integrated simulation specification in this paper. However, there is no *I2T* transformation considered in [4–6]. This paper generalizes and extends the results from [4–6] 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 results have been used to analyze an *S2T* transformation of a production system (a domain-specific visual model) to Petri nets. 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 [10].

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. 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 [15], nowadays widely used for model transformation specification.

References

1. OMG: Unified Modeling Language: Superstructure – Version 2.1.1. (2005) formal/07-02-05, http://www.omg.org/technology/documents/formal/uml.htm.
2. Reisig, W.: Petri Nets. Volume 4 of EATCS Monographs on Theoretical Computer Science. Springer Verlag (1985)
3. Narayanan, A., Karsai, G.: Using Semantic Anchoring to Verify Behavior Preservation in Graph Transformations. In: Proc. Workshop on Graph and Model Transformation (GraMoT'06). Volume 4., Electronic Communications of the EASST (2006)
4. 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)
5. Ermel, C., Ehrig, H., Ehrig, K.: Semantical Correctness of Simulation-to-Animation Model and Rule Transformation. In: Proc. International Workshop on Graph and Model Transformation (GraMoT'06), Satellite Event of the IEEE Symposium on Visual Languages and Human-Centric Computing. Volume 4 of Electronic Communications of the EASST., Brighton, UK, European Association of Software Science and Technology (2006)
6. Ermel, C., Ehrig, H.: Behavior-preserving simulation-to-animation model and rule transformation. In König, B., Heckel, R., Rensink, A., eds.: Proc. of Workshop on Graph Transformation for Verification and Concurrency (GT-VC'07). Volume 213 of ENTCS., Elsevier Science (2008) 55–74
7. Ehrig, H., Ermel, C.: Semantical Correctness and Completeness of Model Transformations using Graph and Rule Transformation. In: Proc. International Conference on Graph Transformation (ICGT'08). Volume 5214 of LNCS., Heidelberg, Springer Verlag (2008) 194–210
8. de Lara, J., Vangheluwe, H.: Translating Model Simulators to Analysis Models. In Fiadeiro, J., Inverardi, P., eds.: Proc. Fundamental Approaches to Software Engineering (FASE'08). Volume 4961 of Lecture Notes in Computer Science. (2008) 77–92
9. Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed Graph Transformation with Node Type Inheritance. Theoretical Computer Science **376**(3) (2007) 139–163
10. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theoretical Computer Science. Springer Verlag (2006)
11. Parisi-Presicce, F.: Transformation of Graph Grammars. In: 5th Int. Workshop on Graph Grammars and their Application to Computer Science. Volume 1073 of LNCS., Springer (1996)
12. Kreowski, H.J.: A Comparison between Petri Nets and Graph Grammars. In: 5th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 100, Springer (1981) 1–19
13. Ehrig, H., Ehrig, K.: Overview of Formal Concepts for Model Transformations based on Typed Attributed Graph Transformation. In: Proc. International Workshop on Graph and Model Transformation (GraMoT'05). Volume 152 of ENTCS., Tallinn, Estonia, Elsevier Science (2005)
14. Varró, D.: Automated formal verification of visual modeling languages by model checking. Software and System Modeling **3**(2) (2004) 85–113
15. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In Tinhofer, G., ed.: WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science. Volume 903 of Lecture Notes in Computer Science., Heidelberg, Springer Verlag (1994) 151–163